VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NAVIGACE MOBILNÍHO ROBOTU B2 VE VENKOVNÍM PROSTŘEDÍ

NAVIGATION OF B2 MOBILE ROBOT IN OUTDOOR ENVIRONMENT

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. DAVID HOFFMANN**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. JIŘÍ KREJSA, Ph.D.**

SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. David Hoffmann**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**doc. Ing. Jiří Krejsa, Ph.D.**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Navigace mobilního robotu B2 ve venkovním prostředí

Stručná charakteristika problematiky úkolu:

Navigace mobilního robotu se skládá ze základních dílčích úloh, kterými jsou lokalizace, tedy určení polohy v prostředí, a plánování, a to plánování globální s ohledem na cíl a plánování lokální, které souvisí s detekcí a vyhýbání se překážkám. Mobilní robot B2, určený pro venkovní prostředí, je v současné době schopen úlohu navigace zvládnout pouze do jisté míry. Podstatou diplomové práce je zlepšit současné výkony robotu úpravou lokalizačních a plánovacích subsystémů robotu, případně úpravou hardware.

Cíle diplomové práce:

Seznamte se s frameworkem ROS a mobilním robotem B2.
Proveďte sérii testů autonomního pohybu v reálném prostředí.
Kvantifikujte kvalitu navigace robotu B2 vyhodnocením provedených testů.
Navrhněte, implementujte, verifikujte a vyhodnoťte úpravu lokalizačních a plánovacích subsystémů robotu pro zlepšení kvality autonomního pohybu robotu B2.

Seznam doporučené literatury:

KORYTÁR Lukáš, Realizace lokalizačního systému pro mobilní robot B2, Brno, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018

ROS.org. ROS.org | Powering the world's robots. [online]. 2.11.2016 [cit. 2016-11-02]. Dostupné z: http://www.ros.org/

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt
Diplomová práce se zabývá navigací mobilního robotu, který využívá framework ROS. Cílem je zlepšení schopnosti autonomního pohybu existujícího robotu B2 ve venkovním prostředí. Teoretická část obsahuje popis jádra navigace, které tvoří knihovna *move_base* a balíčky použité pro plánování. Praktická část dokumentuje nedostatky odhalené u dosavadního řešení, návrh a implementaci změn a výsledky z následného testování v prostředí městského parku.

Summary
This master's thesis deals with the navigation of a mobile robot that uses the ROS framework. The aim is to improve the ability of the existing B2 robot to move autonomously outdoors. The theoretical part contains a description of the navigation core, which consists of the *move_base* library and the packages used for planning. The practical part describe the flaws of the existing solution, the design and implementation of changes and the results of subsequent testing in the urban park environment.

Klíčová slova
ROS, Robot Operating System, navigace, autonomní robot, robot B2

Keywords
ROS, Robot Operating System, navigation, autonomous robot, robot B2

HOFFMANN, D. *Navigace mobilního robotu B2 ve venkovním prostředí.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2019. 67 s. Vedoucí diplomové práce doc. Ing. Jiří Krejsa, Ph.D.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Jiřího Krejsy, Ph.D. a s použitím zdrojů uvedených v seznamu literatury.

Bc. David Hoffmann

Děkuji svému vedoucímu doc. Ing. Jiřímu Krejsovi, Ph.D. za ochotu a přátelský přístup během vedení této práce. Dále děkuji rodičům a všem blízkým za pomoc a podporu během celého studia.

Bc. David Hoffmann

Obsah

1. Úvod

Tématem této práce je navigace mobilního robotu ve venkovním prostředí. Jinými slovy jde o dosažení schopnosti autonomního pohybu například v prostředí městského parku, kdy robot na základě senzorických dat a uložené mapy dokáže bez zásahu obsluhy naplánovat cestu a následně dojet do cílové pozice.

Důkazem o aktuálnosti této problematiky v dnešní době je snaha prakticky všech významných firem v automobilovém průmyslu o vytvoření autonomních vozidel. Současná veřejnosti dostupná řešení jsou zatím zpravidla spíše velmi pokročilými asistenty řízení a stále vyžadují nepřetržitou kontrolu člověkem, ať už z důvodů technických nebo právních. Faktem nicméně je, že jsou vyvíjeny automobily, které by v budoucnu měly dokázat v běžném provozu zcela nahradit člověka v roli řidiče. Například společnost Tesla nedávno představila nový počítač FSD (*Full Self-Driving*) [1], který údajně disponuje 21x větším výkonem než dosavadní řešení od společnosti NVIDIA a měl by, jak napovídá název, stačit pro potřeby plně autonomního řízení.

V rámci práce je využíván framework ROS (*Robot Operating System*) [2]. Jedná se o software s open source licencí BSD, na jehož počátku stála snaha Stanford University vytvořit dynamický systém určený pro využití v robotice. Tuto snahu v roce 2007 podpořila společnost Willow Garage [3], která pro účely vývoje poskytla nemalé prostředky a také nespočet výzkumníků, kteří přispěli svým časem a znalostmi jak při vývoji jádra ROSu, tak i tvorbou různých balíčků. V současnosti čítá komunita kolem ROSu několik desítek tisíc uživatelů po celém světě a je využíván jak v komerční sféře, tak pro hobby projekty. Od svého vzniku prošel původní ROS množstvím změn a dnes je k dispozici již dvanáctá distribuce s názvem Melodic Morenia. Jedná se o verzi s prodlouženou podporou do roku 2023. O popularitě ROSu svědčí také například vznik jeho odnože ROS-Industrial [4], který je vhodnější pro potřeby průmyslu, nebo vývoj jeho nové generace ROS 2 [5].

Přesto, že tato práce předpokládá u čtenáře alespoň základní znalost frameworku ROS, budou zde pro jistotu vysvětleny alespoň dva základní termíny. První z nich je *node*, což je v ROSu označení programu nebo procesu plnícího určitou funkci (např. plánování, zpracování senzorických dat, lokalizace atd.). *Node* může být typu *Subscriber* (přijímá data), *Publisher* (odesílá data), případně (*Publisher/Subscriber*). Pro označení streamů dat, ke kterým se jednotlivé *nody* připojují za účelem publikování nebo čtení, slouží termín *topic*. Protože by byl překlad těchto i některých dalších termínů zavádějící a komplikoval by jejich případné vyhledávání, je v této práci používáno originální znění v anglickém jazyce. Rozsáhlá dokumentace a návody týkající se ROSu jsou dostupné na ROS wiki [6]. Stručnější popis v českém jazyce obsahuje diplomová práce [7].

Cílem této práce je konkrétně zlepšení navigačních schopností mobilního robotu B2, čímž navazuje na diplomovou práci [8], která se zabývala především lokalizací a navigace zde byla řešena pouze okrajově. Proto je část této práce věnována analýze výchozího stavu robotu B2. Dále jsou zde popsány provedené úpravy, které byly navrženy za účelem eliminace nalezených nedostatků, testování schopnosti autonomní jízdy zejména v oblasti křižovatek a následné vyhodnocení. Teoretickou část tvoří popis navigačních nástrojů frameworku ROS použitých v robotu B2.

2. Mobilní robot B2

Mobilní robot B2 je robot s Ackermannovým podvozkem určený pro jízdu ve venkovním prostředí. Robot byl původně vytvořen v roce 2007 jako projekt FSI VUT, v roce 2009 se účastnil robotické soutěže Robotour. Následně byl robot B2 využit v několika závěrečných pracích a v roce 2015 prošel úpravou HW [8]. Konstrukční úpravy drobnějšího charakteru pak byly dále provedeny v roce 2018. Současná podoba robotu odpovídá obr. 2.1.

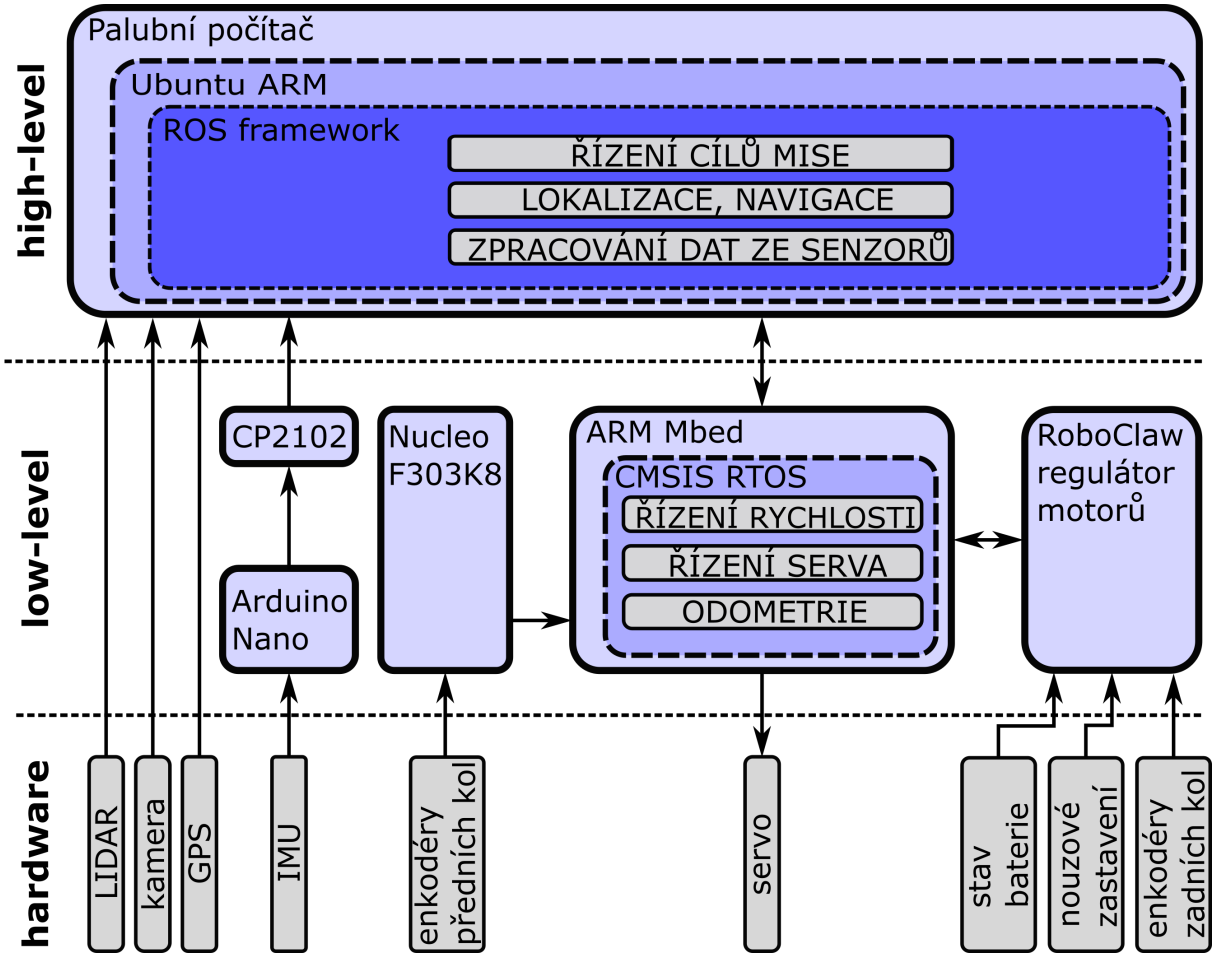
Obrázek 2.1: Robot B2

Řízení robotu B2 je rozděleno do dvou úrovní a jeho schéma je zachyceno na obr. 2.2. *Low-level* řízení běží na mikrokontroleru ARM Mbed. Ten ovládá motory, řídí natočení kol, čte data z enkodérů a v případě zmáčknutí STOP tlačítka zodpovídá za nouzové zastavení. Dále pak komunikuje s *high-level* řízením a to pomocí komunikačního protokolu, který je popsán v bakalářské práci [9].

High-level řízení obstarává jednodeskový počítač, na kterém je nainstalován operační systém ARM Ubuntu a framework ROS, konkrétně distribuce Indigo [10]. Na tomto počítači běží lokalizační a navigační systém robotu a také je využíván ke zpracování senzorických dat. Konkrétní senzorická výbava robotu vypadá takto:

- Lidar SICK-LMS200
- Kamera Microsoft LifeCam HD-3000

- IMU jednotka MPU-9150
- GPS Navilock NL-302U
- Odometrický modul



Obrázek 2.2: Topologie řídicí části [8]

Jak již bylo zmíněno, robotem B2 se zabývalo několik bakalářských a diplomových prací a je zbytečné zde dále přepisovat jejich obsah. I přes to, že zejména starší z nich nejsou již vzhledem k později provedeným změnám zcela aktuální, velká část jejich obsahu je stále platná. Za poslední čtyři roky jsou to závěrečné práce [9, 11, 12, 8].

3. Navigace

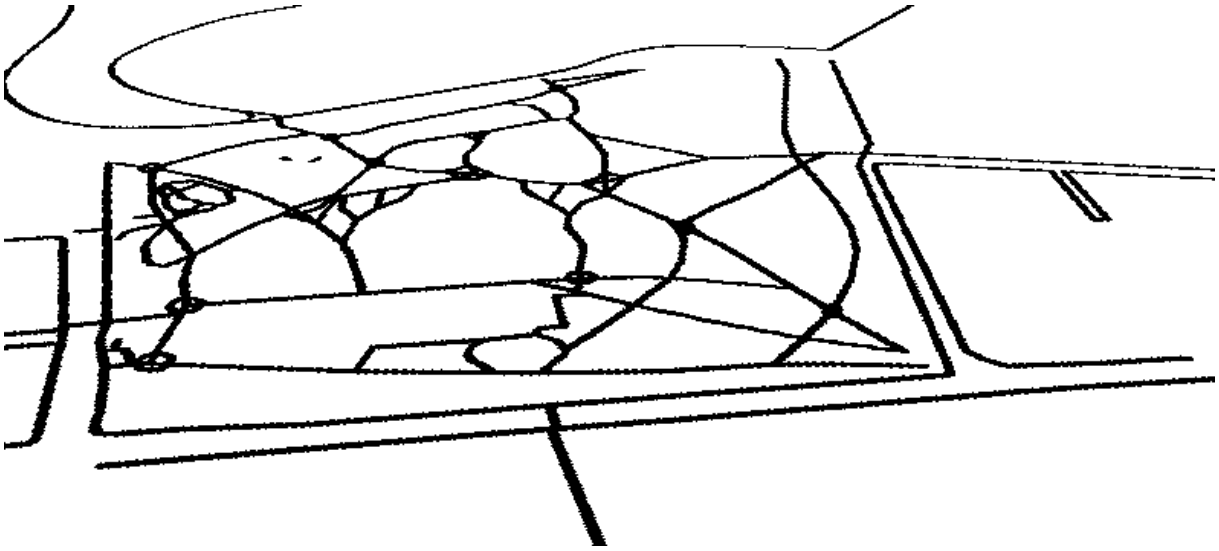
Klíčovými úkoly navigace jsou lokalizace a plánování trasy. V ROSu existuje pro tyto účely mnoho volně dostupných nástrojů. Při výběru vhodných balíčků je však třeba zohlednit, že některé z nich nejsou vzájemně kompatibilní. Nástroje pro navigaci sice byly vybrány již v DP [8], která se zabývala lokalizací, avšak během tvorby této práce byla otázka zmíněné volby znovu otevřena a bylo zjištěno, že tato volba zásadně závisí na reprezentaci mapy, kterou robot používá pro popis prostředí, v němž se pohybuje. Možnosti reprezentace mapy popisuje detailněji kapitola 3.1.

3.1. Reprezentace mapy

Byly nalezeny tři možnosti jak reprezentovat mapu způsobem použitelným pro navigaci. Výhody a nevýhody jednotlivých řešení jsou popsány v následujících podkapitolách. Shrnutí získaných poznatků a volbě nástrojů pro navigaci se pak věnuje kapitola 3.2.

3.1.1. Bitmapa

V tomto případě je mapa reprezentována obrázkem ve formátu podporovaném knihovnou SDL [13]. Balíček, který slouží jako zprostředkovatel takovéto mapy pro ROS, je *map_server* [14]. Ten navíc potřebuje kromě obrázku s mapou soubor ve formátu *.yaml*, který obsahuje potřebná metadata jako název souboru obrázku, počátek mapy, její orientaci a rozlišení. Příklad mapy reprezentované pomocí bitmapy zobrazený v prostředí RViz zachycuje obr. 3.1. Takováto reprezentace má v případě velkých map nevýhodu. Je totiž třeba použít malé rozlišení, aby nebyl obrázek příliš velký. Zde konkrétně odpovídá 1 px ploše 1 m^2. Další nevýhoda této reprezentace mapy se projevuje při hledání globálního plánu cesty. Pixely určující šířku cesty zde totiž nemají praktický význam a pouze zpomalují výpočet. Tento problém však není tak zásadní, neboť globální plán není zpravidla nutné příliš často aktualizovat.

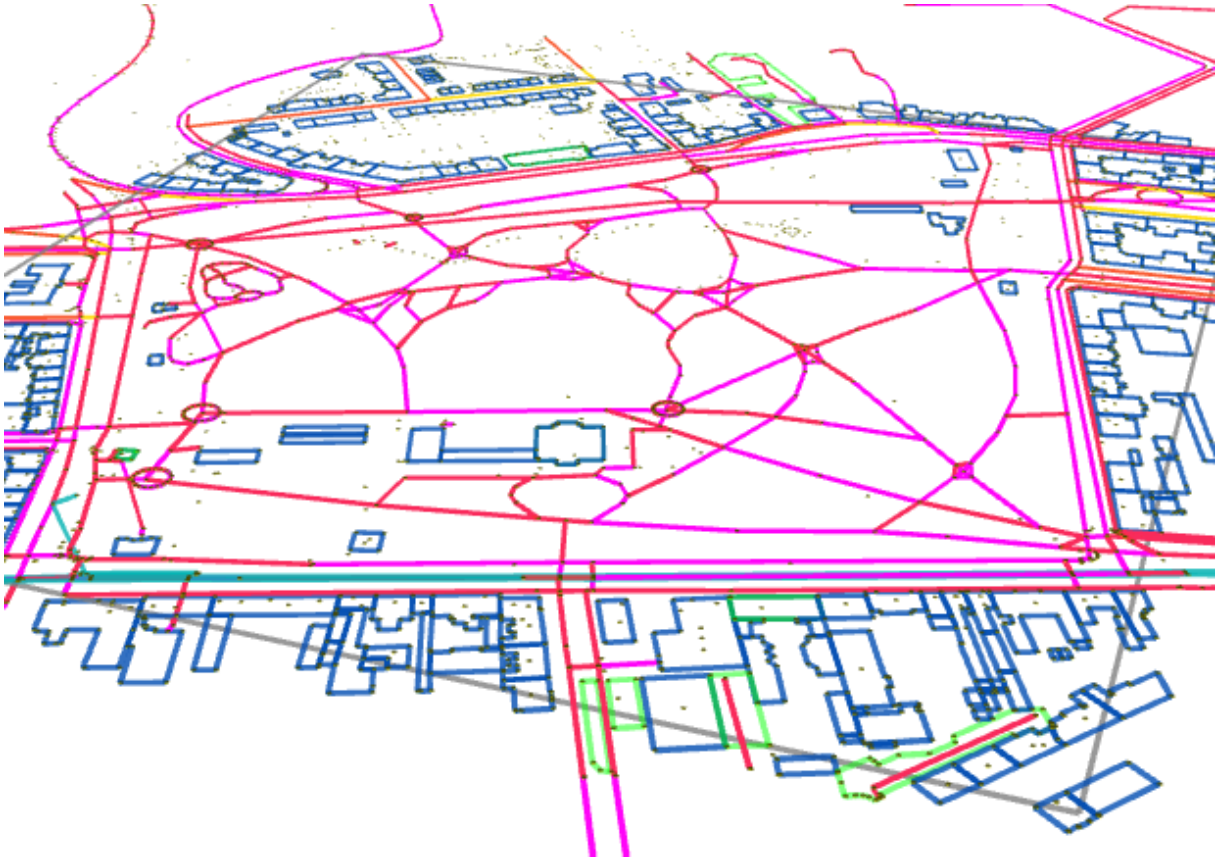


Obrázek 3.1: Mapa reprezentovaná bitmapou

Výše byly zmíněny především nevýhody tohoto řešení, nicméně má i zásadní pozitiva. Jeho největší výhodou je fakt, že *map_server* je přímo součástí *2D navigation stacku* [15], což znamená, že je tato reprezentace mapy použitelná pro všechny balíčky kompatibilní s tímto souborem knihoven. Nezanedbatelnou výhodou je také skutečnost, že je tato mapa použitelná jak ke globálnímu, tak v případě potřeby i k lokálnímu plánování a je pro ni vyřešena lokalizace.

3.1.2. Open Street Maps

Tato reprezentace mapy využívá soubor *.osm*, který lze získat exportem přímo ze stránek projektu *Open Street Maps* [16]. Pro použití této mapy v ROSu slouží balíček *osm_cartography* [17]. Vizualizaci v RVizu zachycuje obr. 3.2. Mapa je v tomto případě zobrazena jako *vizualization_marker_array* [18] a jde o její plnou podobu (je možné vytvořit filtrovanou mapu pouze s cestami).



Obrázek 3.2: Mapa ve formátu OSM

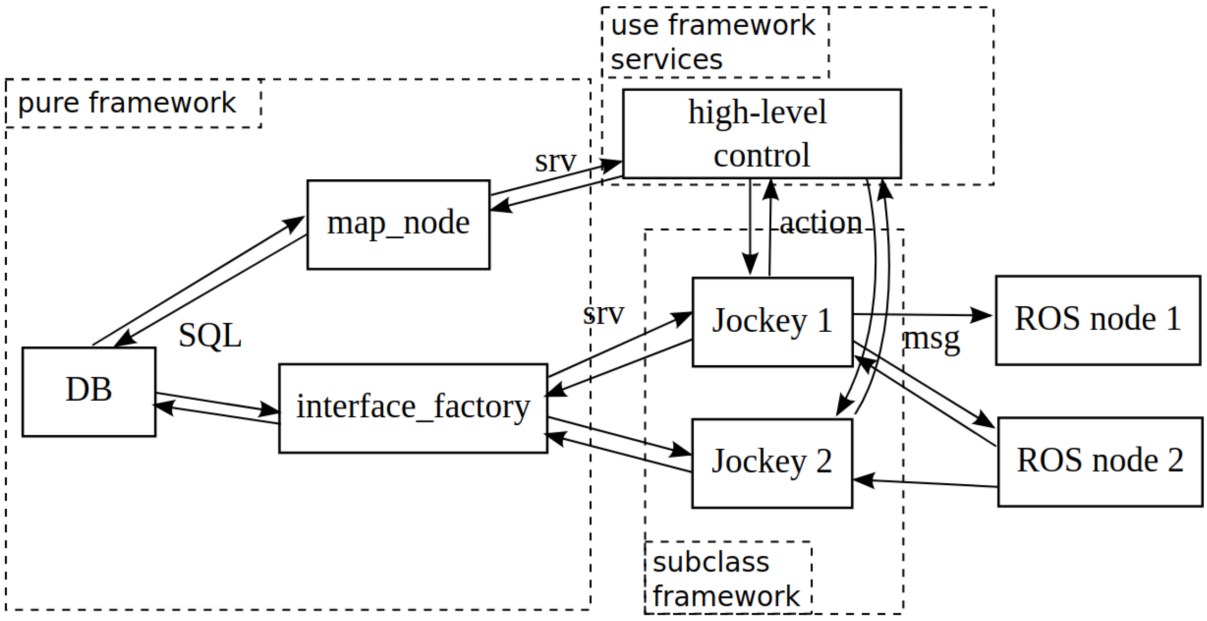
Pro plánování cesty v takovéto mapě existuje balíček *route_network* [19]. Jedná se však pouze o experimentální balíček s poslední úpravou v roce 2012. Dále byl nalezen balíček *osm_planner* [20] a také se plánováním v takto reprezentované mapě zabývala bakalářská práce [21] z Fakulty informatiky VUT.

I přes to, že se tato reprezentace mapy jeví pro větší mapy jako výhodnější, než ta uvedená v kapitole 3.1.1, ukázalo se, že nalezené plánovače mají špatnou dokumentaci, práce s nimi je komplikovaná a slouží pouze k nalezení globálního plánu. Případnou potřebu mapy pro lokální plánování, kdy je tento typ zcela nevyhovující, by tedy bylo nutné

vyřešit jiným způsobem. Navíc by při použití tohoto typu mapy bylo třeba provést značné zásahy do již existují lokalizace.

3.1.3. SQL databáze

Mapu uloženou do databáze používá framework *LaMa* (*Large Maps*) [22]. Strukturu mapy tvoří orientovaný graf, který je ukládán do SQL databáze v podobě hran a uzlů. Těm jsou dále přiřazovány tzv. deskriptory, jimiž jsou například délka cesty nebo data ze senzorů jako laser scan. V případě velké mapy se jedná o praktičtější řešení podobně jako u předchozí varianty.



Obrázek 3.3: Schema frameworku *LaMa* [22]

Framework *LaMa* je modulární systém, který je tvořen ze třech částí (obr. 3.3). První část nazvaná *pure framework* je rozhraní mezi SQL databází a *nody* využívajícími framework. Část *subclass framework* obsahuje základní třídy, které umožňují implementaci tzv. *jockeys*. Těch existují tři typy. Pro učení, pro lokalizaci a pro navigaci. Poslední, třetí část frameworku *use framework services*, obstarává *high-level* řízení, do něhož spadá správa mapy a zadávání cílů *jokeys*.

Na rozdíl od řešení uvedených v předchozích podkapitolách obsahuje *LaMa* nástroje pro vytvoření mapy, lokalizaci i navigaci v této mapě. I přes to bylo použití tohoto frameworku vyhodnoceno jako nejméně vhodné. Jak je zmíněno také v práci [8], před tím, než je možné použít navigaci, je nutné projet konkrétní místo s robotem pomocí ovladače, aby si vytvořil mapu. (Předchozí dvě řešení popsaná v kapitolách 3.1.1 a 3.1.2 používají mapu staženou z internetu, což je elegantnější). Lokalizace v takto vytvořené mapě bude navíc téměř jistě mnohem citlivější na změny prostředí, neboť je výrazně vyšší pravděpodobnost, že se změní okolí cesty popsané deskriptory než cesta samotná. Další nevýhodou je dostupnost *LaMa* pouze pro distribuci *ROS Indigo*, kterému v době psaní této práce končí podpora.

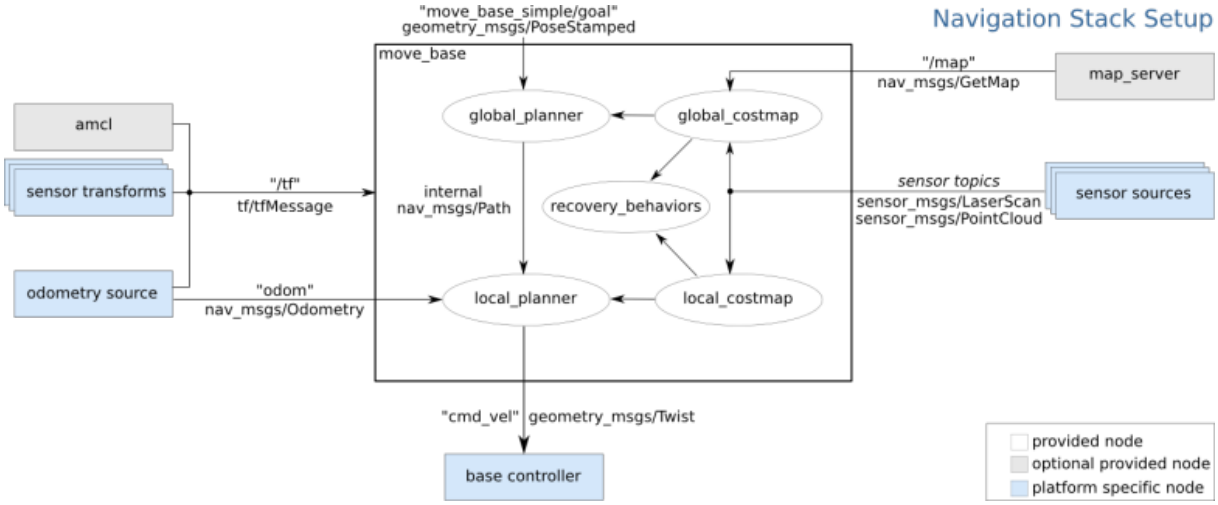
3.2. Shrnutí a výběr nástrojů pro navigaci

Po zvážení výhod a nevýhod prozkoumaných možností byla potvrzena volba použít pro reprezentaci mapy bitmapu a jako jádro navigace balíček *move_base*, kterému se věnuje kapitola 3.3. Jako plánovače jsou použity *global_planner* a *teb_local_planner* popsané v podkapitolách 3.3.2 a 3.3.3. Toto rozhodnutí bylo učiněno na základě prakticky stejných důvodů, jaké jsou uvedeny v práci [8]. Těmi jsou konkrétně nejkvalitnější dokumentace a kompatibilita s mnoha dalšími balíčky.

Byla zvažována také možnost využití mapy ve formátu *.osm*. Toto řešení by sice přímo nevylučovalo využití *move_base* a jeví se jako elegantnější a praktičtější než bitmapa, avšak implementace by byla velmi náročná a přínos z hlediska funkčnosti minimální. Využití frameworku *LaMa* bylo na základě nevýhod popsaných v kapitole 3.1.3 zavrženo.

3.3. Move base

Move_base [23] je balíček patřící do *navigation stacku*, který tvoří jádro navigace a umožňuje fungování navigační úlohy jako celku. Princip fungování *move_base* zachycuje obr. 3.4



Obrázek 3.4: Schema fungování *move_base* [23]

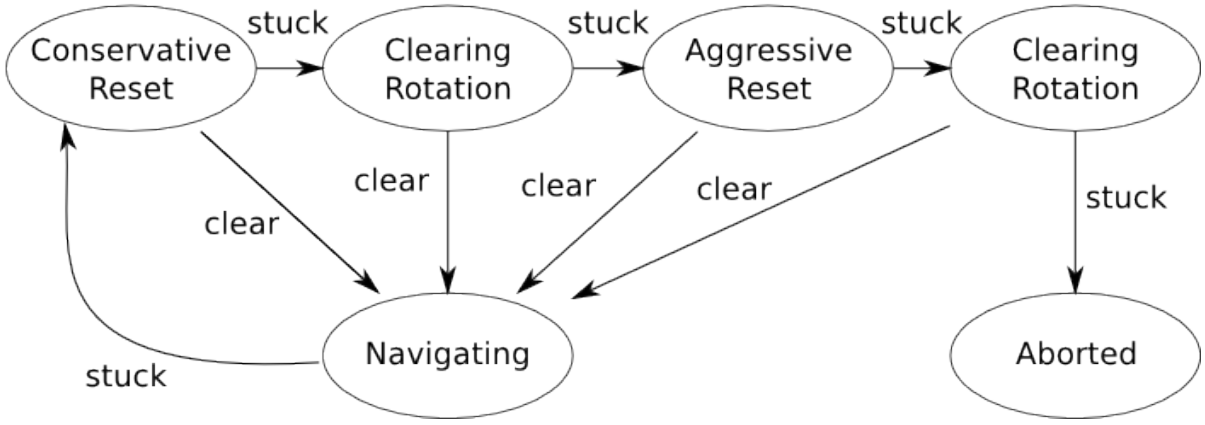
Z obrázku je patrné, že vstupy *move_base* jsou údaje o mapě, data ze senzorů, odometrická data, transformace mezi souřadnicovými systémy, údaje z lokalizace a požadovaný cíl. Na základě těchto vstupů pak *move_base* vytváří plán cesty a publikuje data sloužící pro řízení robotu. Plánování cesty probíhá ve dvou krocích. Nejprve je vytvořen globální plán (cesta mezi počáteční polohou a zadaným cílem) a ten je poté sledován lokálním plánovačem. Lokální plánovač by přitom měl být schopen reagovat na dynamické prostředí a zohledňovat omezení robotu (kinematika, limity rychlosti, zrychlení...). Výstupem z lokálního plánovače jsou údaje o aktuálně požadovaných zobecněných rychlostech robotu (translační, úhlová).

V souboru knihoven *navigation* je k dispozici několik plánovačů. Z globálních jsou to jmenovitě *carrot_planner* [24], *navfn* [25] a *global_planner* [26], z lokálních pak *base_local_planner* [27] a *dwa_local_planner* [28]. Mimo *navigation stack* jsou dostupné i mnohé

další s ním kompatibilní plánovače jako například globální *voronoi_planner* [29] nebo *eband_local_planner* [30] a *teb_local_planner* [31] pro lokální plánování.

Kromě plánovačů spravuje *move_base* také mapy překážek (*costmaps*). Každé pole v této mapě má na základě hodnotící funkce přiřazenu "cenu", přičemž překážky a jejich okolí jsou penalizovány. Tato penalizace zajišťuje, aby nalezená cesta nevedla ke kolizi. Podrobnějšímu popisu costmap se věnuje kapitola 3.3.1.

Může se stát, že robot chybně detekuje překážky ve svém okolí a takzvaně "uvízne". Pro tento případ je součástí *move_base* nástroj *recovery_behaviors*, který slouží k vyčištění costmapy. Výchozí nastavení nástroje pro zotavení costmapy vystihuje schéma na obr. 3.5. V případě potřeby lze pomocí parametrů postup pro zotavení upravit. Je totiž například zřejmé, že v případě robotu s Ackermannovým podvozkem, není možné využít *Clearin Rotation*.



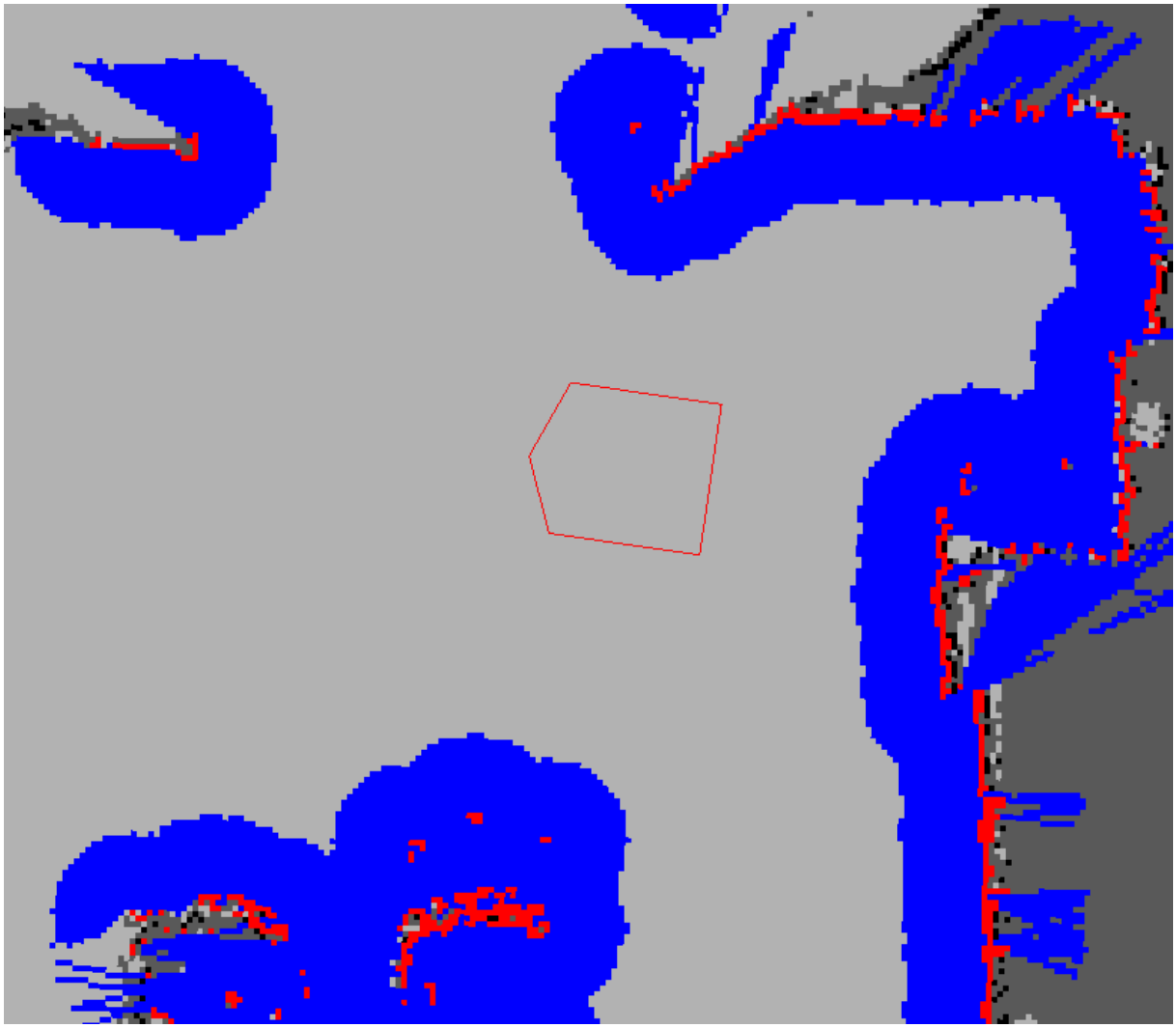
Obrázek 3.5: Zotavení costmapy [23]

Pro konfiguraci *move_base* slouží řada parametrů. Pomocí nich lze zvolit plánovače, nastavit *recovery_behaviors*, frekvenci s jakou má pracovat řídicí smyčka, čas po kterém mají být spuštěny nástroje pro zotavení, nebo s jakou frekvencí má pracovat globální plánovač. Pro aplikace, kde jsou možnosti *move_base* nedostačující, je možné použít *move_base_flex* [32].

3.3.1. Costmapy

Termín costmap slouží v ROSu jako označení pro "mapu překážek" v okolí robotu. Obvykle bývají použity 2 costmapy, jedna pro globální plánování, druhá pro lokální. Pro vytváření costmap je v souboru knihoven *navigation* implementován balíček *costmap_2d* [33]. Poloha překážek je do costmapy zaznamenávána na základě senzorických dat. I přes to, že název navozuje dojem využití pouze ve 2D, umožňuje tento balíček pracovat ve 3D (viz *voxel_grid* [34]). Pro inicializaci costmapy je možné využít mapu poskytovanou *map_serverem* a její velikost může být v případě potřeby omezena pouze na okolí robotu (tzv. *"rolling window"*). Příklad costmapy zachycuje obr 3.6, kde červené body reprezentují buňky obsahující překážku, modře jsou vykresleny oblasti inflace (okolí překážky se zvýšenou "cenou" buněk) a červený polygon značí obrys robotu (footprint).

Může být matoucí, že parametr *footprint* figuruje též v parametrech *teb_local_planneru*. Zde je však pouze pro účely optimalizace a může být odlišný od parametru costmapy. Pro kontrolu proveditelnosti naplánované trajektorie je využíván parametr costmapy [35].



Obrázek 3.6: Costmapa zobrazená v prostředí RViz []

Od verze *Hydro* je možné (a však ne nutné) rozdělit pomocí plug-inů costmapu na funkční vrstvy. Základem jsou vrstvy *Static Map Layer*, *Obstackle Map Layer* a *Inflation Layer*, ale mohou být přidány i další (*Social Layer*,*Range Sensor Layer*). Protože je správné nastavení costmap naprosto klíčové pro zajištění správné funkce plánovačů, budou následující řádky věnovány detailnějšímu popisu jednotlivých vrstev.

Static Map Layer

Static Map Layer [36] je vrstvou costmapy, která obsahuje převážně neměnná data z externích zdrojů (typicky *map_server*). Parametry této vrstvy lze nastavit, například název topicu, do kterého je publikována mapa, jaká cena má být přiřazena neprozkoumanému prostoru, má-li být mapa pravidelně aktualizována a zda má být v této vrstvě použita celá škála hodnot nebo stačí rozlišovat stavy *No information*, *Free space* a *Lethal obstackle.*

Obstackle Map Layer

Vrstva *Obstackle Map Layer* [37] sdružuje informace ze senzorů ve formě *PointCloud* [38] nebo *LaserScan* [39] a uchovává informace o poloze překážek. Vzhledem k tomu, že se

robot může nacházet v dynamickém prostředí, případně může dojít k chybnému určení překážky, je žádoucí, aby byly překážky do této vrstvy nejen zapisovány, ale také mazány. K tomu slouží práva *marking* a *clearing*, pomocí nichž lze nastavit, zda může konkrétní senzor překážky přidávat, odebírat nebo obojí.

Parametry v této vrtvě costmapy lze rozdělit podle jejich funkce do několika skupin. Podstatné jsou především ty ze skupin *Sensor Management Parameters* a *Global Filtering Parametres*.

Skupina *Sensor Management Parameters* obsahuje parametr *observation_sources*. Ten umožňuje zvolit, jaké "zdroje pozorování" chceme použít. Další parametry slouží k nadefinování těchto zdrojů. Přiřadí jim jméno (vstup pro parametr *observation_sources*), příslušný topic, jehož data mají být použita, o jaký typ dat se jedná, práva a maximální vzdálenost pro *marking*/*clearing*, maximální a minimální výšku překážek a další.

Global Filtering Parametres obsahuje pouze tři parametry. První z nich se jmenuje *max_obstacle_height* a jeho hodnota by měla být vyšší, než je výška robotu. Další parametry jsou *obstacle_range*, který udává maximální vzdálenost pro *marking* a *raytrace_range*, který udává maximální vzdálenost pro *clearing*. Tyto dva parametry mohou být přepsány parametrem konkrétního senzoru.

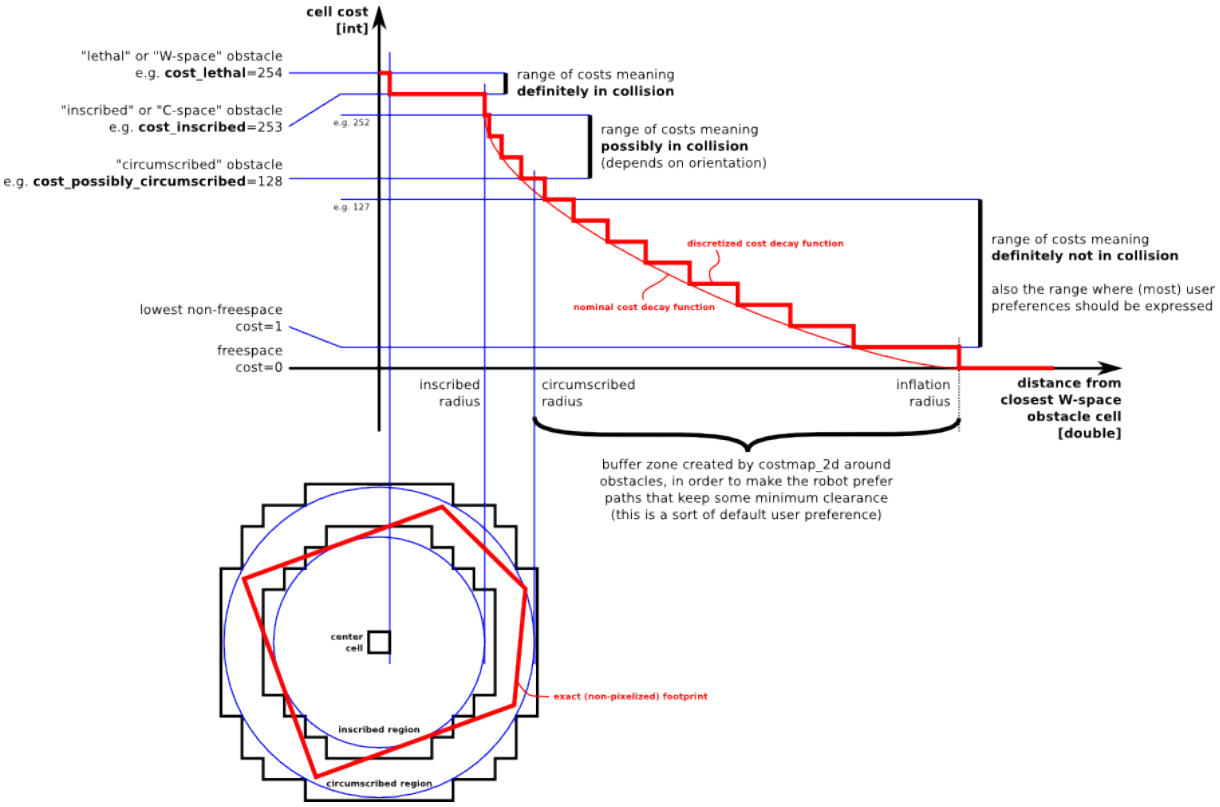
Inflation Layer

Ve vrstvě *Inflation Layer* [40] je jednotlivým buňkám přiřazována "cena", která klesá s rostoucí vzdáleností od překážky. Tato cena může nabývat hodnot $0 - 254$ (255 je překážka) a její rozložení dobře vysvětluje obr. 3.7. Pro popis této vrstvy bylo definováno 5 kategorií, do kterých je možné buňky na základě ceny rozdělit.

- "*Lethal cost*" je cena buňek v takovém okolí překážky, kde se robot definitivně nachází v kolizi.
- "*Inscribed cost*" je horní hranice ceny, při níž se robot již nemusí nutně nacházet v kolizi. Záleží však na jeho natočení vzhledem k překážce.
- "*Possibly circumscribed cost*" je cena, pod jejíž hranicí by při správném nastavení nemělo dojít ke kolizi.
- "*Freespace*" je označení pro buňky s nulovou cenou.
- "*Unknown*" slouží pro zařazení neprozkoumaných buněk. Cenu těchto buněk může nastavit uživatel podle potřeby.

Cena hodnotící funkce je v rozsahu mezi *inscribed_radius* a *inflation_radius* počítána podle vztahu 3.1 [40], kde C je hodnota parametru *cost_scaling_factor*, D odpovídá vzdálenosti od překážky, R je *inscribed_radius* a I *costmap_2d::INSCRIBED_INFLATED_OBSTACLE*, což odpovídá hodnotě 254. Vzhledem k tomu, že je hodnota *cost_scaling_factor* násobena záporným číslem, zvyšování hodnoty tohoto parametru snižuje výslednou cenu.

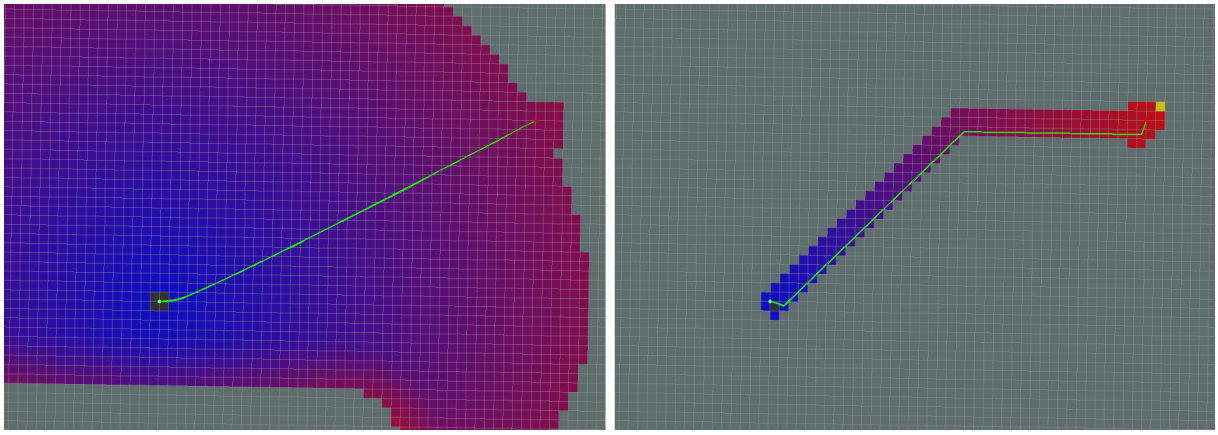
$$cost = exp(-1 \cdot C \cdot (D - R)) \cdot (I - 1) \tag{3.1}$$



Obrázek 3.7: Rozložení cen buněk okolo překážek [40]

3.3.2. Globální plánovač

Pro použití v robotu B2 byl zvolen globální plánovač *global_planner* [26], který je součástí *navigation stacku*. Tento plánovač vyhledává cestu v potenciálovém poli, k čemuž jsou zde implementovány dva algoritmy. Defaultním algoritmem je Dijkstra, druhou možností A*. Ten je však implementován s Manhattanskou metrikou [41]. To má za následek, že nalezená cesta nemusí být optimální. Porovnání těchto algoritmů zachycuje obr. 3.8.



Obrázek 3.8: Porovnání algoritmů Dijkstra (vlevo) a A* [26]

Kromě typu použitého algoritmu je možné nastavit řadu dalších parametrů. Za zmínku stojí především parametry *neutral_cost* a *cost_factor*. Jejich hodnoty figurují ve výpočtu cen jednotlivých buněk potenciálového pole a tím ovlivňují tvar nalezené cesty. Cena

políčka je určena podle vztahu 3.2 [42]. Vlivu velikosti hodnot těchto parametrů na tvar plánu se podrobněji věnuje článek [43].

$$cost = cost_{neutral} + cost_{factor} \cdot costmap_{cost\_value} \tag{3.2}$$

Další parametry globálního plánovače jsou:

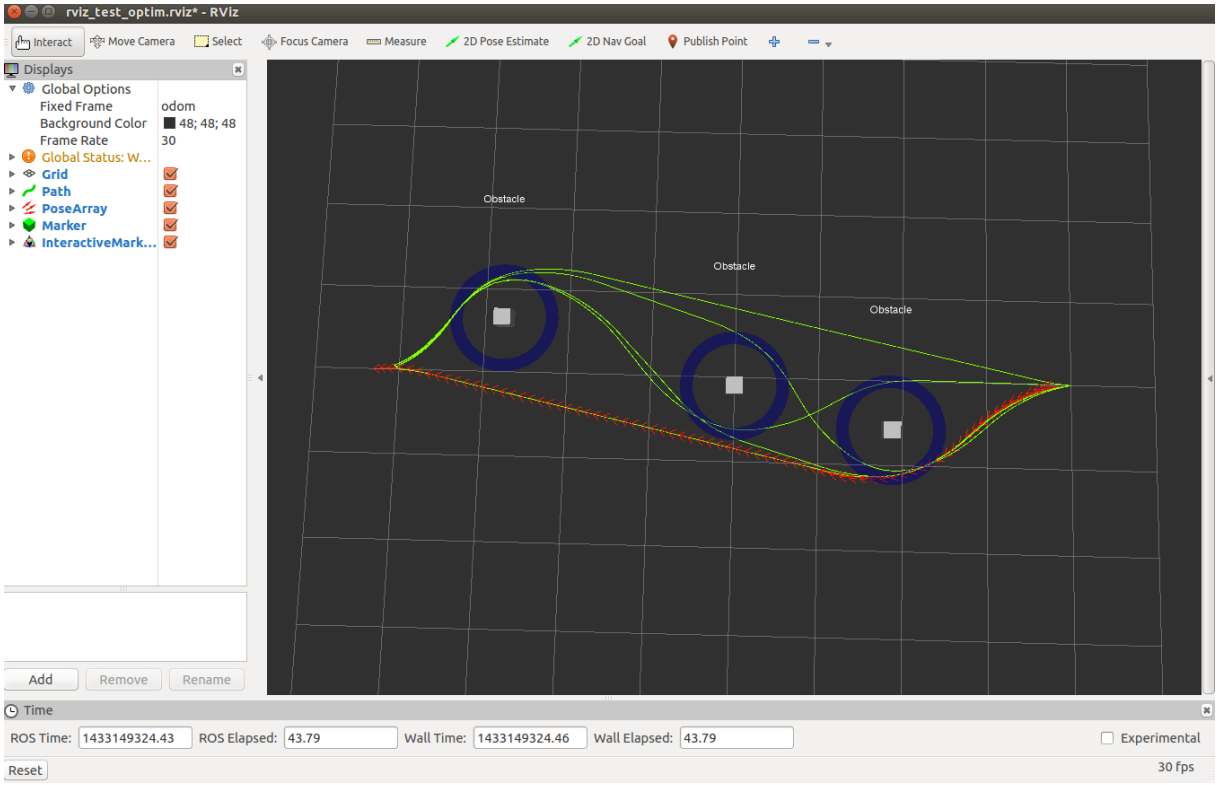
- *allow_unknown* (bool, default: true) - Specifikuje, zda smí plánovač vytvořit plán, který vede přes neznámý prostor. Zde je třeba dát pozor při použití vícevrstvé 2D costmap, kde je zapotřebí nastavit parametr *track_unknown_space* jako *true*, jinak je všechen neznámý prostor považován za čistý.
- *default_tolerance* (double, default: 0.0) - Maximální dovolená vzdálenost nalezeného koncového bodu plánu od zadaného cíle.
- *visualize_potential* (bool, default: false) - Specifikuje, zda má být potenciálové pole zobrazeno ve formě 2D point cloudu.
- *use_quadratic* (bool, default: true) - Je-li nastaveno jako *true*, je potenciál aproximován kvadratickou funkcí. V opačném případě je výpočet jednodušší.
- *use_grid_path* (bool, default: false) - Je-li nastaveno jako *true*, nalezený plán sleduje mřížku. Defaultně je použita metoda gradientního sestupu.
- *old_navfn_behavior* (bool, default: false) - Pro případ kdy je požadováno chování plánovače totožné s *navfn*. Ostatní booleovské parametry je třeba nastavit na defaultní hodnoty.
- *lethal_cost* (int, default: 253) - Cena, při jejímž překročení nedojde k nalezení plánu.
- *publish_potential* (bool, default: true) - Specifikuje, zda má být publikováno potenciálové pole.
- *orientation_mode* (int, default: 0) - Určuje orientaci v každém bodě plánu. (None=0, Forward=1, Interpolate=2, ForwardThenInterpolate=3, Backward=4, Leftward=5, Rightward=6)

3.3.3. Lokální plánovač

Vhodným lokálním plánovačem pro použití v robotu B2 je *teb_local_planner* [31]. Tento plánovač využívá metodu nazvanou *Timed Elastic Band* a je schopen plánovat trajektorie pro neholonomní roboty (od verze Kinetic podporuje i holonomní roboty), přičemž respektuje jejich kinematická a dynamická omezení. Velkou výhodou tohoto plánovače je kompatibilita s *2D Navigation stackem*.

Příklad trajektorie nalezené pomocí plánovače *teb_local_planner* zachycuje obr. 3.9. Z tohoto obrázku je patrné, že plánovač hledá paralelně více trajektorií a následně vybere nejlepší z nich.

Chování *teb_local_planneru* je možné ovlivnit nastavením řady parametrů. Ty je možné rozdělit do několika skupin: konfigurace robotu, cílová tolerance, konfigurace trajektorie, překážek, optimalizace, charakteristika topologie plánování a jiné. Protože je těchto parametrů výrazně větší množství, než v případě *global_planneru*, budou zde popsány pouze výše zmíněné skupiny.



Obrázek 3.9: Teb local planner [44]

Konfigurace robotu

Do této skupiny parametrů patří údaje o robotu, jako jsou limit rychlosti a zrychlení, maximální úhlová rychlost a zrychlení, minimální poloměr zatáčení a rozvor. Také je zde možné zvolit, aby byla úhlová rychlost ve výstupu z plánovače nahrazena odpovídajícím úhlem natočení kol. Při povolení této možnosti dochází ve standardně používané zprávě typu *twist* ke změně významu rychlosti otáčení. V tomto ohledu poskytuje lepší popis zpráva typu *ackerman_msgs* [45], nicméně tento typ zpráv není podporován knihovnou *move_base*.

Tolerance dosažení cíle

Tyto parametry umožňují nastavit maximální dovolenou velikost chyby dosažení cíle v metrech a chybu natočení v radiánech. Poslední parametr umožňuje odstranit požadavek na rychlost v cílovém bodě.

Konfigurace trajektorie

Pomocí parametrů trajektorie lze nastavit její rozlišení, minimální počet bodů, povolit práva pro přepis orientace v jednotlivých bodech globálního plánu, nastavit maximální délku globálního plánu pro optimalizaci nalezené trajektorie nebo také např. za jakých okolností má být trajektorie přepočítána při změně předchozího cíle. Jednou z možností je také nastavení zpětné vazby, která obsahuje celou trajektorii a výpis aktivních překážek. Tato možnost by však měla být aktivována jen pro účely ladění.

Parametry překážek

Tyto parametry slouží k nastavení minimální požadované vzdálenosti plánu od překážky, zda má výpočet brát v úvahu překážky z costmapy a jaká vzdálenost od překážek je při plánování znevýhodněna. V případě, že je použit *costmap_converter* [46], je možné nastavit, zda má být costmapa převedena na body, úsečky nebo polygony. Dále je možné přidělit výpočtu této konverze samostatné vlákno a nastavit frekvenci s jakou má být aktualizována.

Parametry optimalizace

Zde jsou přiděleny váhy některým dříve zmíněným parametrům, které jsou využívané při optimalizaci trajektorie. Například jak moc má robot preferovat jízdu vpřed, jak je důležité dodržet minimální vzdálenost od překážky nebo jak moc se musí robot držet globálního plánu.

Charakteristika topologie plánování

Tato skupina parametrů ovlivňuje, jakým způsobem dochází k výběru nejlepší z vypočtených trajektorií, kolik trajektorií je paralelně počítáno, zda mají být pro výpočty použita oddělená vlákna a mají-li být trajektorie optimalizovány současně. Některé z těchto parametrů mají značný vliv na vytížení CPU.

Zbývající parametry

Do poslední skupiny spadají pouze dva a to parametr, který slouží pro zadání topicu odometrie a parametr pro zadání *map_frame*.

4. Analýza původního stavu

Před úpravou robotu B2 byly hodnoceny aspekty popsané v následujících podkapitolách.

4.1. Konstrukce

Po stránce hardwaru byly na robotu B2, jehož výchozí stav zachycuje obr. 4.1, nalezeny různé nedostatky. Některé z nich přispívaly k nespolehlivosti robotu, jiné se týkaly pouze vzhledu. Konkrétní výčet nedostatků je následující:

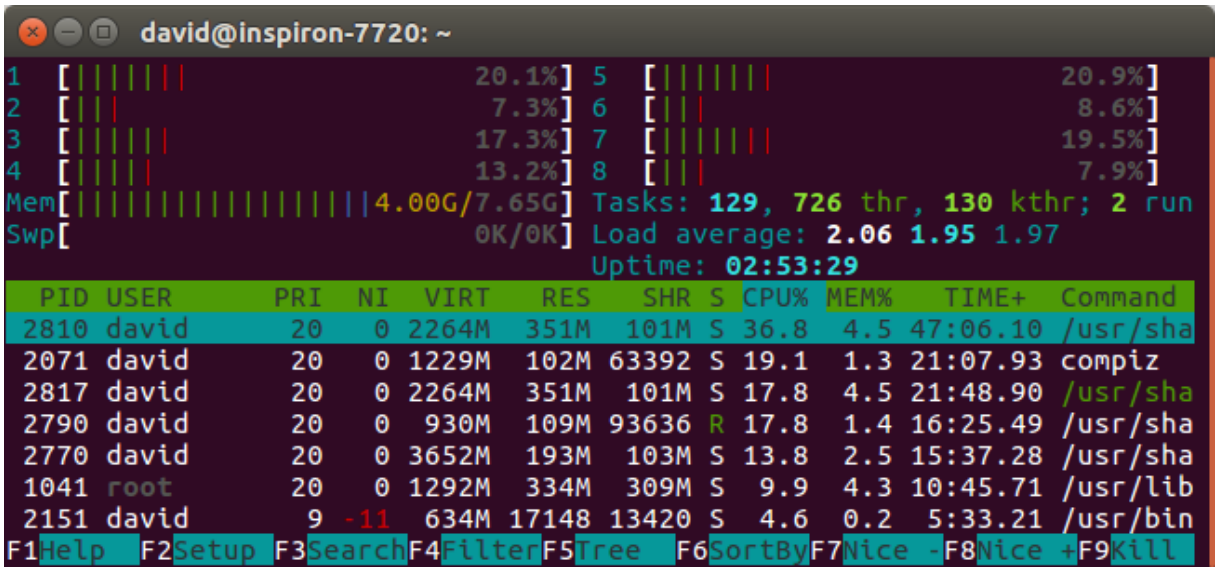
- Špatná tuhost hliníkového profilu s IMU a kamerou
- Špatné uchycení kamery umožňující její natáčení
- Špatně přístupné STOP tlačítko
- Volný konektor napájení scanneru
- Neukotvený USB hub
- Části přilepené lepicí páskou

Obrázek 4.1: Původní vzhled robotu B2

4.2. Detekce překážek

Včasná detekce překážky je nezbytnou podmínkou pro realizaci autonomního pohybu. Během testování však bylo zjištěno, že robot B2 nestíhá dostatečně rychle reagovat a to často ani v případě statických překážek. Hlavní příčinou bylo nadměrné vytížení procesoru počítače při jízdě v autonomním režimu.

Tento problém byl zjištěn s pomocí softwaru pro monitorování systémových prostředků *htop* (obr. 4.2) [47]. Výhodou této aplikace je, že zobrazuje hodnoty přímo v terminálu a je tak možné jednoduše sledovat vytížení počítače v robotu přes ssh protokol bez potřeby použití vzdálené plochy. Nástroj také umožňuje zjistit výpočetní náročnost jednotlivých procesů, což je užitečné při optimalizaci.



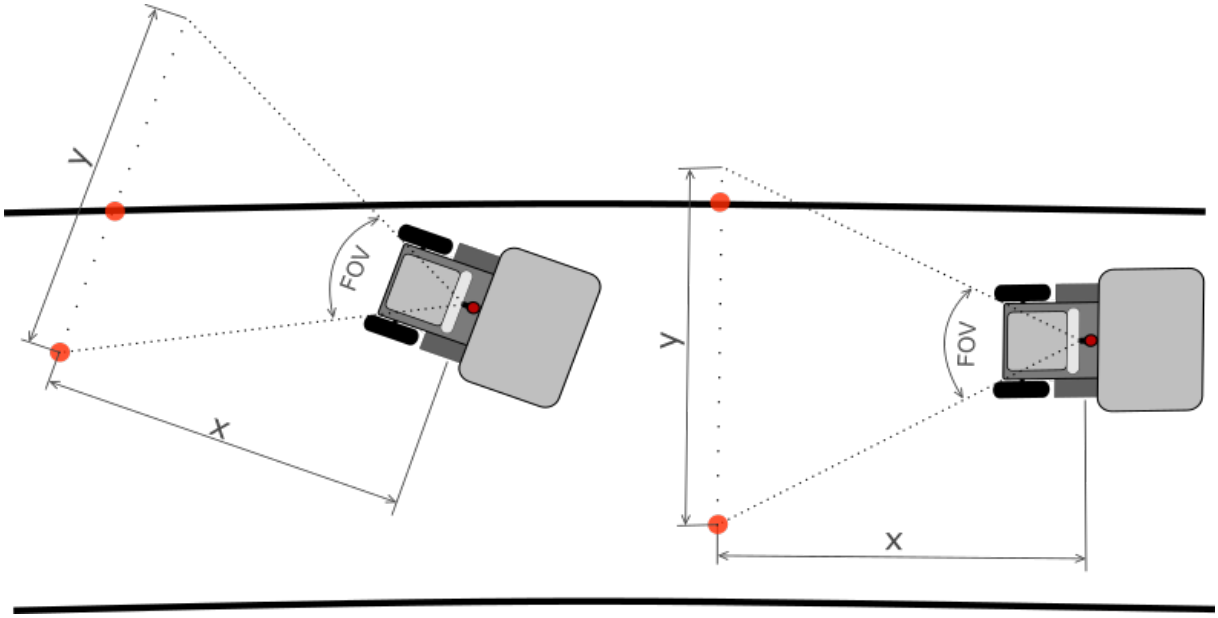
Obrázek 4.2: htop

4.3. Rozpoznávání cesty

K rozpoznávání cesty využívá robot B2 dva zdroje, na základě kterých jsou zvlášť určeny její krajní body. Ty jsou následně fúzovány způsobem popsaným v práci [8]. Jedná se o laserový scanner SICK LMS200 a kameru Microsoft LifeCam HD-3000 . Zde je limitující zorné pole kamery (FOV), které je pro rozlišení 640x480 v horizontálním směru $FOV = 51{,}36\,°$ [8]. Při použité vzdálenosti $x = 3{,}5\,\mathrm{m}$ (obr. 4.3) lze na základě vztahu (4.1) přibližně dopočítat, že šířka pozorovaného pásu je $y = 3{,}4\,\mathrm{m}$.

$$y = tan\left(\frac{FOV}{2}\right) 2x \tag{4.1}$$

Tato šířka je menší, než šířka cest, na kterých byl robot testován. Přesto byla tato vzdálenost pro vyhodnocování ponechána. Pro fúzování by totiž ideálně měly být použity body měřené ve stejné vzdálenosti od robotu. Zvětšení vzdálenosti měření laserovým scanerem však znamená ostřejší úhel mezi povrchem cesty a paprsky scanneru. To má za následek, že každá malá změna sklonu scanneru, případně změna sklonu vozovky způsobí velký rozdíl v měřené vzdálenosti.



Obrázek 4.3: Rozpoznávání obrazu

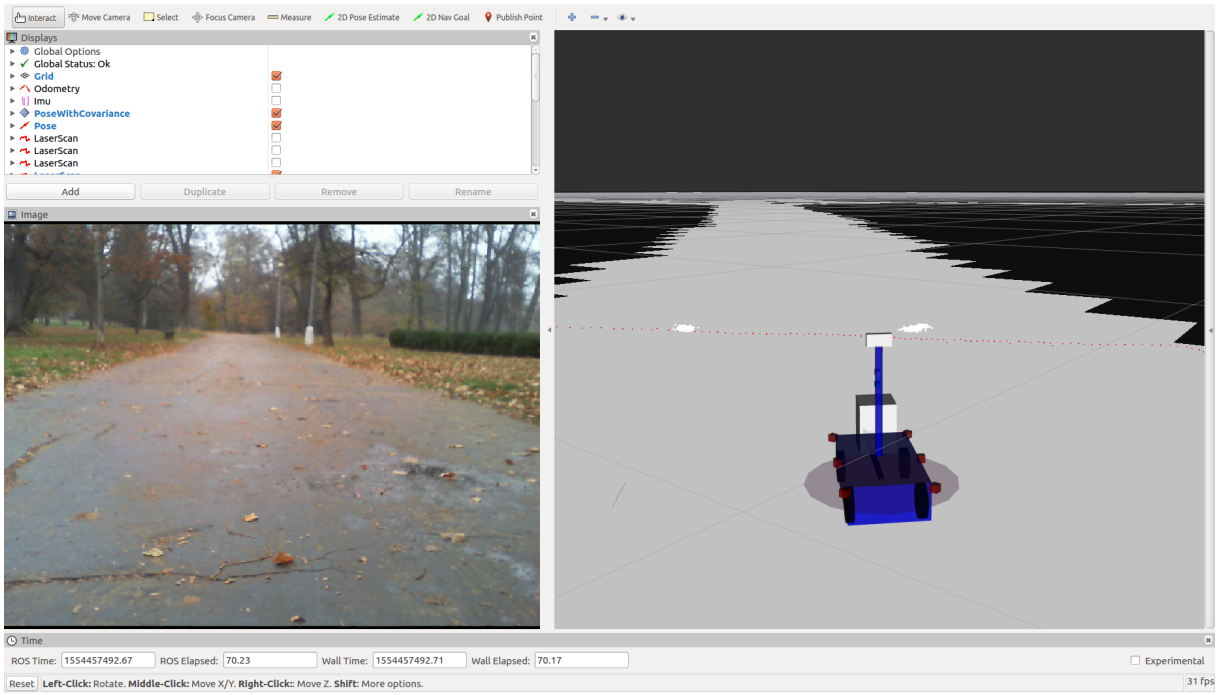
V případě, že robot zachytí pouze jeden kraj cesty, je druhý kraj automaticky doplněn na druhý konec zorného pole. Tuto situaci zachycuje obr. 4.3. Je zřejmé, že se nejedná o problém, je-li robot natočen ve směru vozovky a pohybuje se v blízkosti jejího středu. Levá část obrázku zachycuje situaci, kdy je robot vzhledem k vozovce natočen, případně jede po jejím okraji. To vede v krajním případě až k tomu, že jsou oba krajní body detekovány prakticky ve stejném místě. Za předpokladu dostatečně rovné vozovky byl tento problém do značné míry kompenzován již zmíněnou fúzí krajních bodů cesty, neboť zorné pole lidaru SICK je 180°.

Dále se při vyhodnocování cesty vyskytl problém, kdy algoritmus pro zpracování obrazu vyhodnotil místo skutečných okrajů cesty kontrasty jako např. hrany stínů, mokré fleky či velmi světlé nebo reflexní prvky. V případě vyhodnocování scanu zase docházelo k nepřesnostem v případě větších děr či nerovností cesty. Tento jev však nebyl téměř pozorován, neboť cesty, na kterých byl robot testován, jsou v poměrně dobrém stavu a dominantním problémem byly tedy již zmíněné kontrasty v obraze.

4.4. Uživatelské rozhraní

Pro ovládání ROSu se používá z velké části terminál. Stejně tak je možné v terminálu sledovat data z různých topiců. V mnoha případech je však tato reprezentace dat nepřehledná a je mnohem výhodnější je vizualizovat. K tomuto účelu slouží nástroj RViz (obr. 4.4) [48]. Další výhodou tohoto nástroje je fakt, že umožňuje snadno zadat cíl, do kterého má robot dojet a případně přímo robota ovládat pomocí některého z plug-inů.

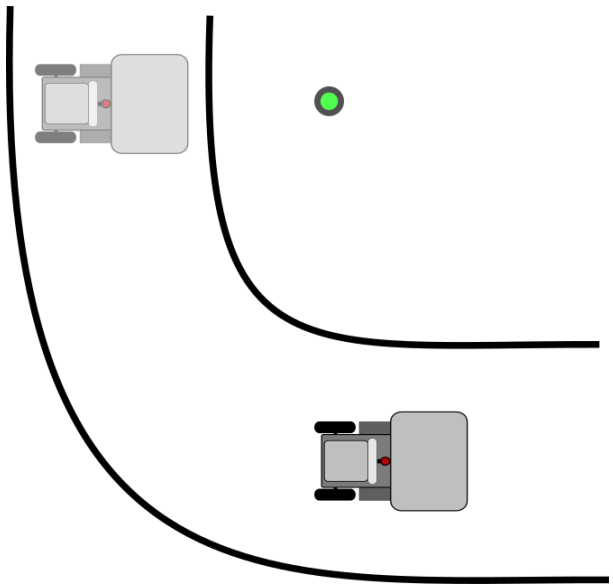
Během úvodních testů byl nalezen prostor pro zlepšení zejména v poskytování informací o tom, jak robot vyhodnocuje obraz z kamery, který je klíčový pro úspěšnou lokalizaci. Další z užitečných informací, které by bylo možné zobrazit v prostředí RVizu jsou také zprávy o tom, kdy si robot ”myslí”, že projíždí křižovatkou, případně informace o stavu baterie.



Obrázek 4.4: Rviz

4.5. Lokalizace

Testováním lokalizace robotu B2, jíž se zabývá diplomová práce [8], byl zjištěn problém s určením počáteční polohy. Ta je získávána na základě GPS, která pro potřeby této aplikace nemá příliš uspokojivou přesnost. Tu navíc ovlivňuje například počasí a hustota zástavby. V souvislosti se špatným určením polohy z GPS byly zaznamenány dva typy problémů.

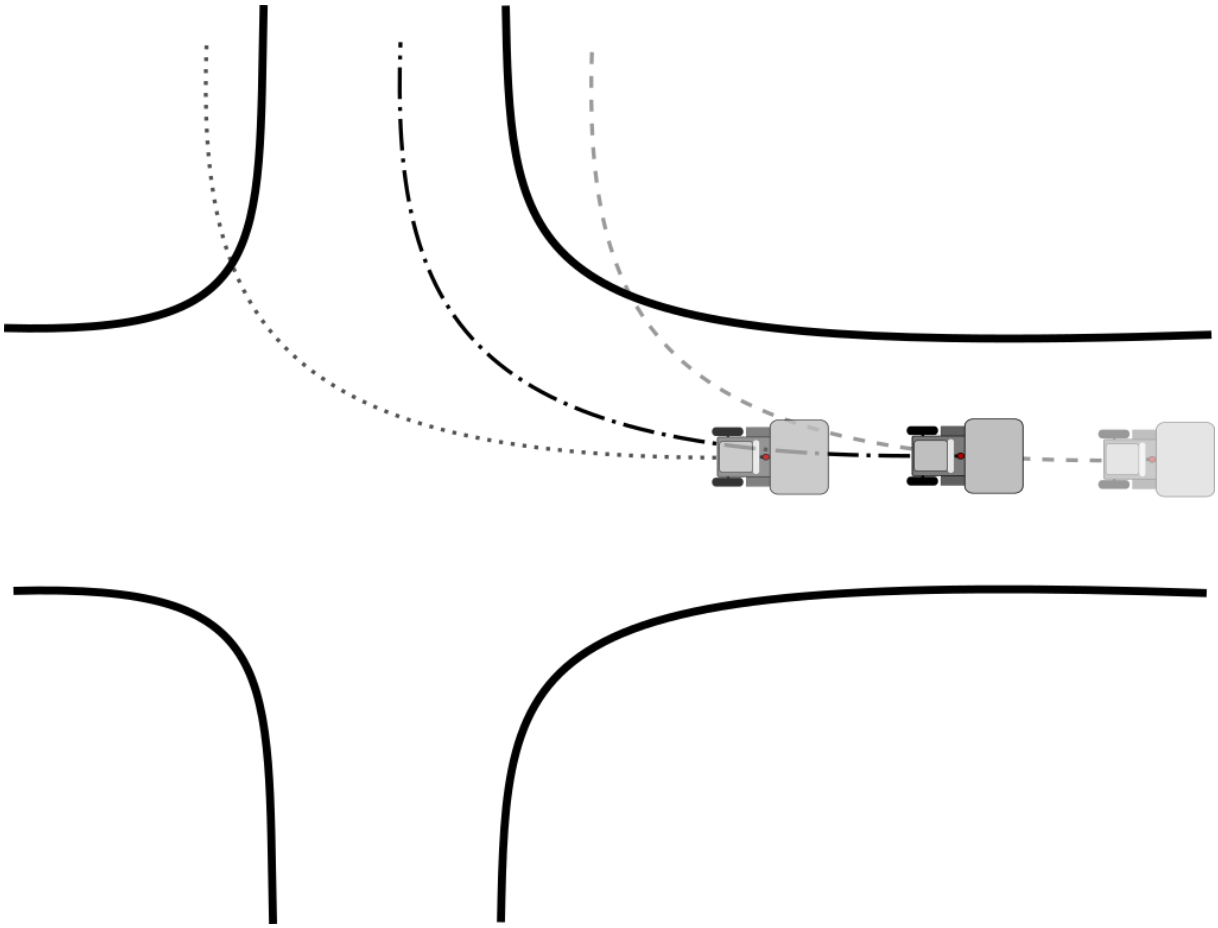


Obrázek 4.5: Chyba lokalizace v zatáčce

První z těchto problémů (obr. 4.5) nemůže nastat v dostatečné vzdálenosti od zatáčky popř. křižovatky. Určení počáteční polohy probíhá tak, že robot hledá nejbližší známý bod

středu cesty. V blízkosti zatáčky se pak může stát, že robot chybně určí svoji počáteční polohu za zatáčkou, nebo se dokonce v blízkosti křižovatky lokalizuje na špatné odbočce.

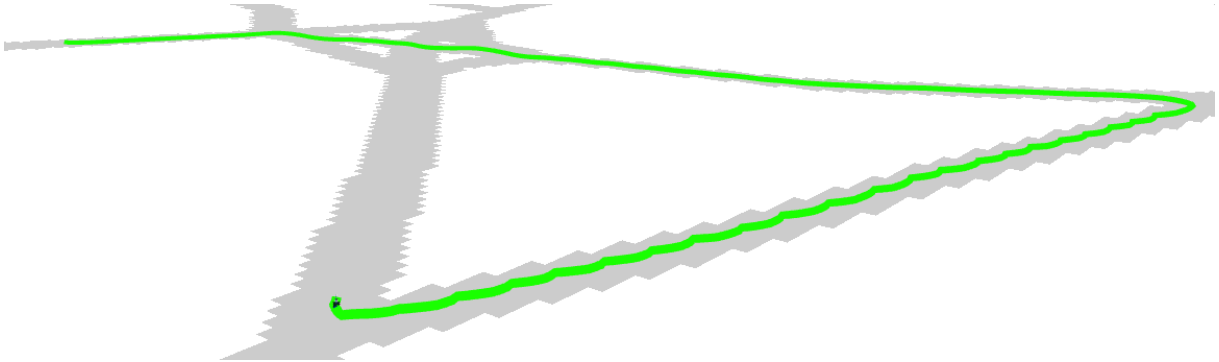
Druhý problém spočívá ve špatné lokalizaci v podélném směru cesty. Během jízdy po rovné cestě nemá robot příliš možnost polohu v tomto směru zpřesnit, neboť je získávána na základě fúze dat z nepřesné GPS a odometrie. Nepřesnost této polohy společně s absencí "křižovatkové logiky", tzn. že robot se v křižovatce lokalizoval pouze na základě dat z IMU a odometrie a jel tzv. "na slepo", pak vedly nezřídka k problému zobrazeném na obr. 4.6, kdy robot zatočil před nebo až za křižovatkou a sjel tak z cesty.



Obrázek 4.6: Chyba průjezdu křižovatkou

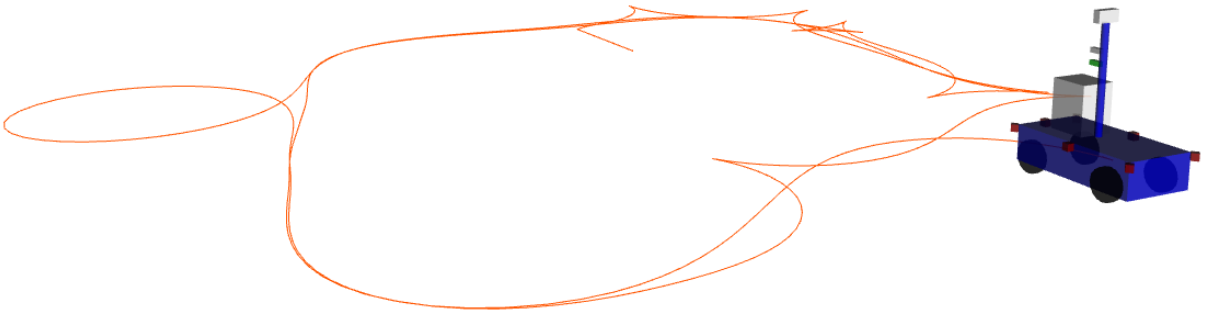
4.6. Plánování trasy

Jako globální plánovač byl použit Base Global Planner [26]. V případě globálního plánu během testovacích měření často docházelo k tomu, že robot nalezl optimální plán cesty, po ujetí několika metrů tento plán přepočítal a snažil se dojet do požadované cílové pozice oklikou a to navíc často couváním, kdy hrozilo zvýšené riziko kolize, neboť robot B2 není ze zadní strany osazen žádnými senzory, které by mohl použít pro detekci překážek. Na globálním plánu se také často vyskytovalo zvlnění. Takový globální plán zachycuje obr. 4.7.



Obrázek 4.7: Globální plán

U lokálního plánu získaného pomocí plánovače Teb Local Planner [31] se navíc vyskytoval problém, kdy robot naplánoval lokální plán zcela nesmyslně. Příklad takovéto špatně naplánované trajektorie zachycuje obr. 4.8.



Obrázek 4.8: Lokální plán

Bylo zjištěno, že hlavním zdrojem problémů obou plánovačů je především špatné nastavení costmap, do kterých byly chybně zapisovány překážky, což v případě globálního plánovače vedlo k přeplánování trasy, neboť původní cestu vyhodnotil jako neprůjezdnou. U lokálního plánovače mělo velké množství překážek zase za následek vytvoření chaotického plánu.

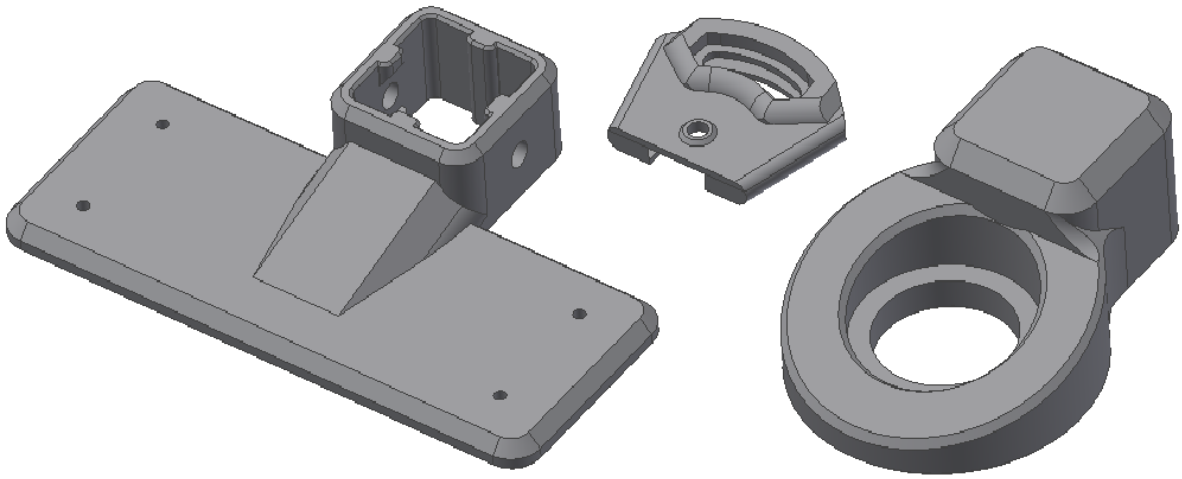
5. Provedené změny

Za účelem zlepšení spolehlivosti jízdy v autonomním režimu byly na základě zjištění popsaných v kap. 4 provedeny úpravy popsané v této kapitole.

5.1. Úprava konstrukce

Konstrukce robotu B2 pochází původně z roku 2007. Od té doby najel robot mnoho kilometrů a prošel mnoha úpravami, které byly v některých případech pouze provizorní a poté nebyl čas na jejich úpravu. Z těchto důvodů byl robot nejprve z velké části rozebrán, vyčištěn a byly promazány pohyblivé části. Poté bylo přikročeno k úpravám konstrukce.

Porovnání výchozího stavu robotu B2 s novým zachycuje obr. 5.4. Vyměněn byl především původní U profil, který sloužil jako držák IMU a kamery a který trpěl značnými vibracemi. Místo tohoto profilu byl použit konstrukční profil Item 20x20. Následně byly navrhnuty a pomocí 3D tisku vyrobeny držák na IMU, kameru a STOP tlačítko (obr 5.1).

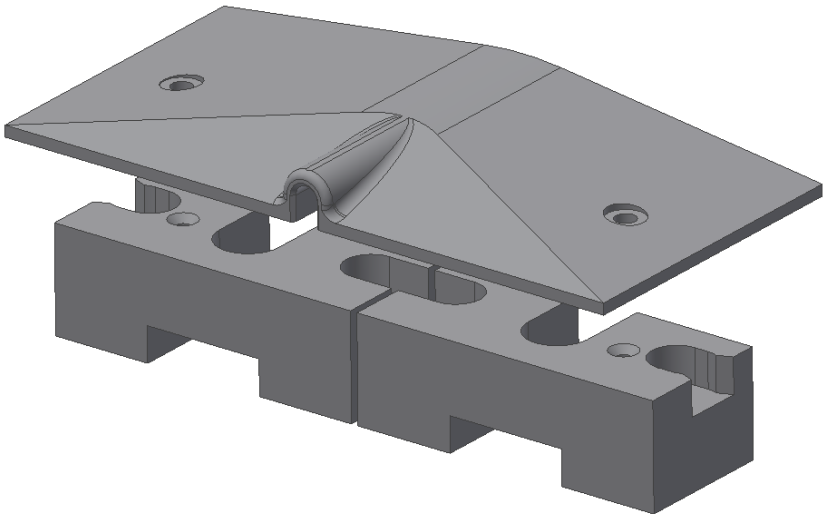


Obrázek 5.1: Držáky IMU, kamery a STOP tlačítka

Držák kamery byl navržen tak, aby na něj přesně lícovala kamera Microsoft LifeCam HD-3000. Původní uchycení totiž dovolovalo její natáčení například vlivem vibrací a po bližším prozkoumání se ukázalo, že šroub, jímž byla kamera zajištěna, držel jen vzepřením o DPS uvnitř kamery. Nový držák kamery drží uvnitř pouze díky zacvaknutí výstupku do dutiny v kameře a navíc je navržen tak, aby bylo možné kameru natáčet v rozsahu cca 5° a v požadované poloze ji zajistit dotažením šroubu.

Další díly byly vytištěny za účelem zajištění konektorů lidaru SICK. Především konektor napájecího kabelu nedržel příliš dobře, a zejména při manipulaci s robotem docházelo nezřídka k jeho neúmyslnému povytažení. Na držáky konektorů byl umístěn kryt, který zároveň slouží k zajištění GPS, která byla do té doby připevněna lepicí páskou. Tyto díly zachycuje obr. 5.2.

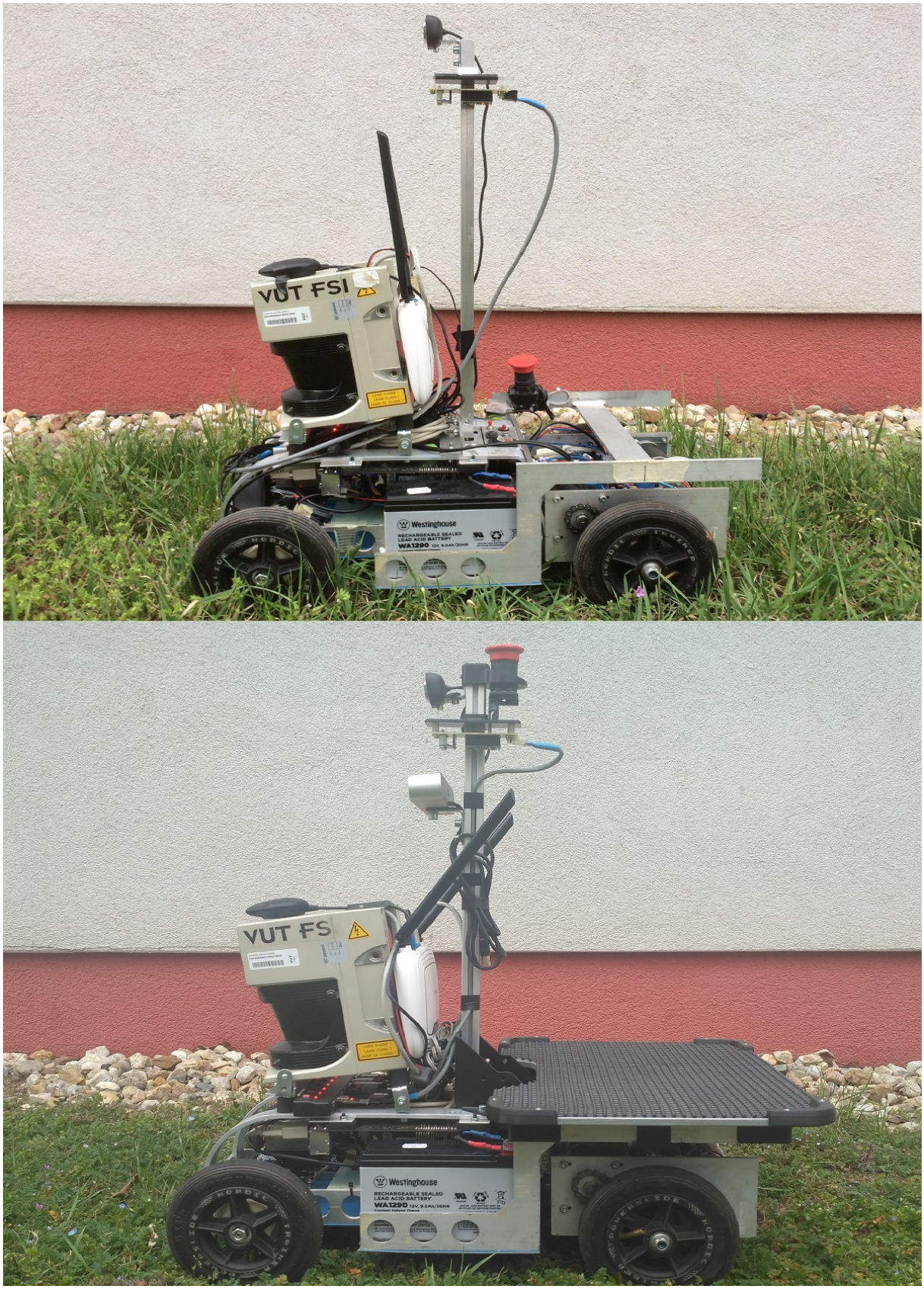
USB hub se volně pohyboval uvnitř robotu a hrozilo tak, že dojde k poškození USB TTL převodníku, pomocí něhož přijímá počítač data z IMU, popř. že dojde k vylomení některého z USB portů. Proto byl hub připevněn k rámu robotu.



Obrázek 5.2: Držák konektorů lidaru a GPS

Poslední provedená úprava na konstrukci je spíše praktická a nemá vliv na funkci robotu. Robot B2 bývá využíván pro různá měření a to i mimo potřeby této práce. Některá z těchto měření vyžadují připojení notebooku kabelem. Z těchto důvodů byl vyroben nosič pro notebook (obr 5.2). Ten není pevnou součástí robotu. Je na něm pouze nasazen a zajištěn dvojicí zpětných zobáčků tak, že je možné ho kdykoli snadno sundat.

Obrázek 5.3: Nosič pro notebook



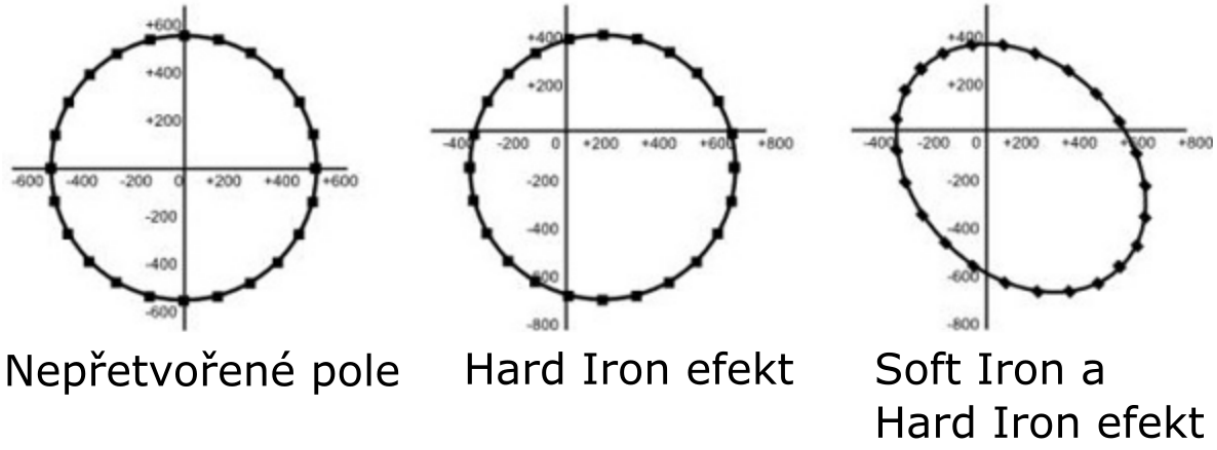
Obrázek 5.4: Porovnání výchozího a nového stavu robotu B2

5.2. Kalibrace

Lokalizační a navigační systém robotu je závislý na senzorických datech. V rámci této práce bylo tedy seřízeno několik senzorů. Tomu se podrobněji věnují následující podkapitoly.

5.2.1. Kalibrace magnetometru

Robot B2 používá IMU modul MPU-9150 [49]. Ten obsahuje tříosý magnetometr, akcelerometr a gyroskop. Data z IMU jsou po zpracování pomocí Madgwickova algoritmu, který je schopen odstranit drift gyroskopu, použita při lokalizaci robotu. K tomu je však nutné, aby byl magnetometer správně seřízen. Toto je popsáno v práci [8], kde jsou také popsány druhy přetvoření magnetického pole.



Obrázek 5.5: Vliv přetvoření magnetického pole [8]

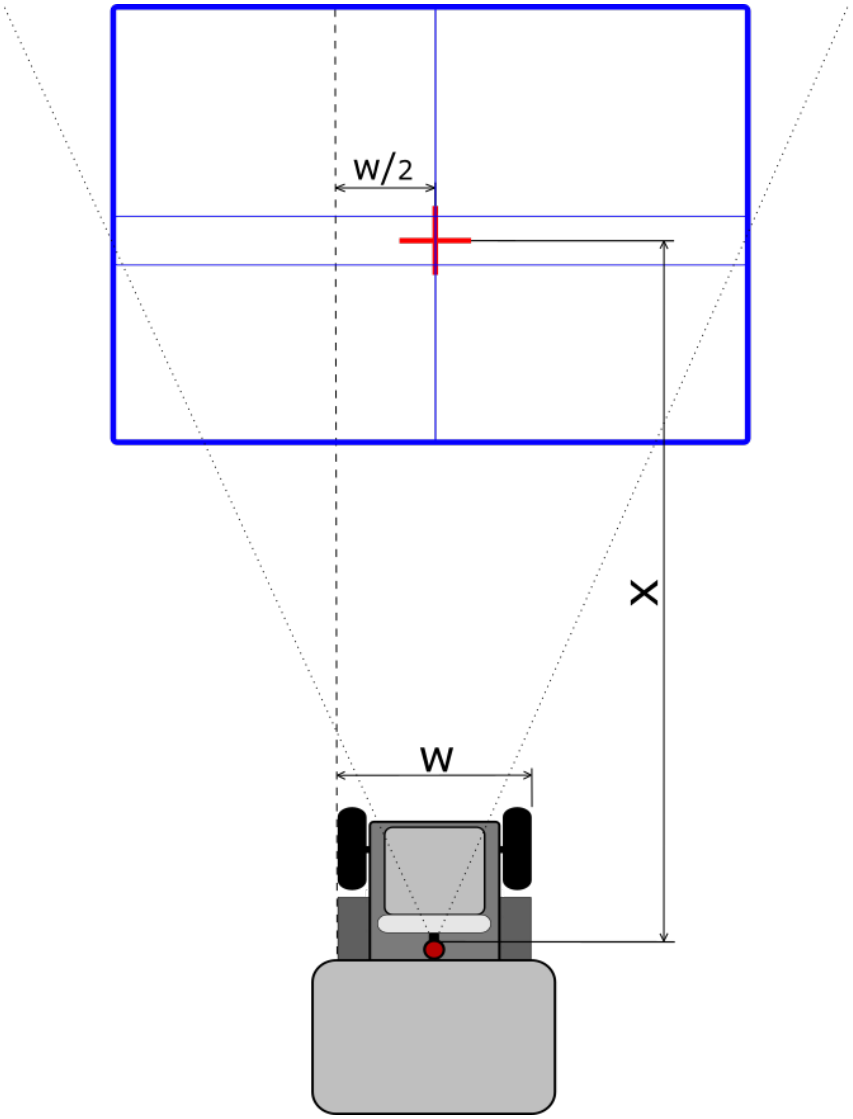
Otočíme-li robotem o 360° kolem svislé osy z a zaznamenáme intenzitu magnetického pole v osách x a y, dostaneme elipsu podobnou té z obr. 5.5. Kalibrace pak prakticky spočívá v nalezení offsetu jejího středu, který dostaneme jako střed minimální a maximální naměřené hodnoty pro danou osu. Tento offset je pak nutné zadat do scriptu *imu_node_4rospack.py*, kde je odečítán od naměřených dat.

U kalibrace magnetometru je potřeba dbát na to, aby byla prováděna v místě, kde nedochází k zakřivení magnetického pole země vlivem okolí robotu. Také je doporučeno provést tuto kalibraci při každé změně hardwaru. Např. GPS anténa umístěná na robotu pro účely jednoho z měření do vzdálenosti cca 30 cm od IMU způsobila nárůst hodnoty intenzity měřeného magnetického pole v daném směru o řád.

5.2.2. Seřízení natočení kamery

Jak bylo zmíněno, uchycení kamery nebylo na původním robotu B2 ideální, neboť nezamezovalo jejímu natáčení a to v rozsahu několika stupňů. Tento nedostatek odstranil nový držák kamery (obr 5.1 uprostřed). Samotné seřízení, které zachycuje také obr 5.6 pak bylo provedeno následovně. Robot byl zarovnán koly podél rovné čáry, do vzdálenosti $x = 3{,}5\,\mathrm{m}$ od kamery a $w/2$ od referenční čáry byla umístěna značka (červeně). Do obrazu z kamery (modře) byla vykreslena svislá osa. Kamera byla poté natočena a zajištěna tak,

aby osa protínala značku. Na závěr byl z obrazu vybrán vodorovný pás používaný při zpracování obrazu a to tak, aby se značka nacházela v jeho středu.



Obrázek 5.6: Seřízení natočení kamery

5.2.3. Seřízení podvozku

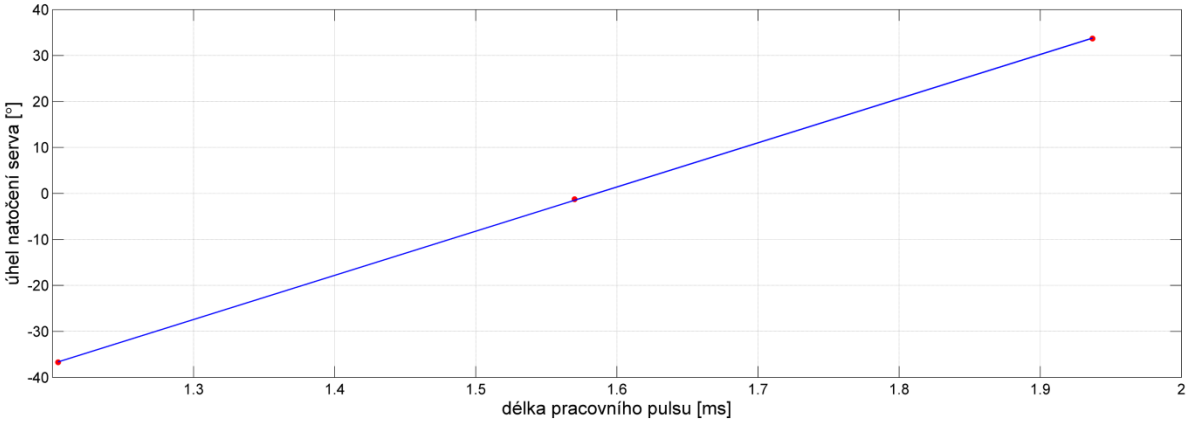
Vlivem četných měření, kdy robot jezdil i po dlažebních kostkách, přejížděl díry ve vozovce a někdy narazil například do obrubníku, došlo k tomu, že robot začal místo jízdy rovně zatáčet a to nezanedbatelným způsobem. Tento jev měl následně velmi nepříznivý vliv na odometrii. Ta je počítána z úhlu natočení předních kol a dat z enkodérů pohonu zadních kol. Detailně se této problematice věnuje diplomová práce [12].

Při výpočtu odometrie konkrétně působily problém dvě věci. Zaprvé robot určuje natočení kol podle hodnoty natočení serva a není nijak měřeno. Druhý problém spočívá v tom, že tato hodnota je používána též v *low-level* řízení jako vstup pro elektronický diferenciál a obzváště na sypaných cestách nebo dlažebních kostkách, kde robot běžně jezdí, pak může docházet k prokluzu kol, což dále znepřesňuje výpočty.

Protože byla zjištěna mírná rozbíhavost předních kol, byly seřízeny tak, aby se naopak mírně sbíhaly a byla tak posílena schopnost jet rovně. Dosáhnout tímto způsobem přesného seřízení je však poměrně náročné a nepraktické. Proto bylo přikročeno k řešení popsanému v následující podkapitole.

5.2.4. Offset serva

Natočení kol robotu B2 je ovládáno modelářským servem. Charakteristika tohoto serva je lineární a zachycuje ji obr. 5.7. Přičtením offsetu k délce řídicího pulzu je možné tuto přímku posunout a tím trimovat natočení kol přední nápravy. Nejprve byl tento offset přičítán v *high-level* úrovni řízení a takto odesílán do *low-level* části. Tím bylo sice dosaženo vyrovnání jízdy robotu, nicméně úprava se velmi negativně projevila na výstupu z odometrie. Při zadání offsetu tímto způsobem totiž ve chvíli, kdy robot jel skutečně přibližně rovně, elektronický diferenciál počítal s tím, že robot zatáčí.

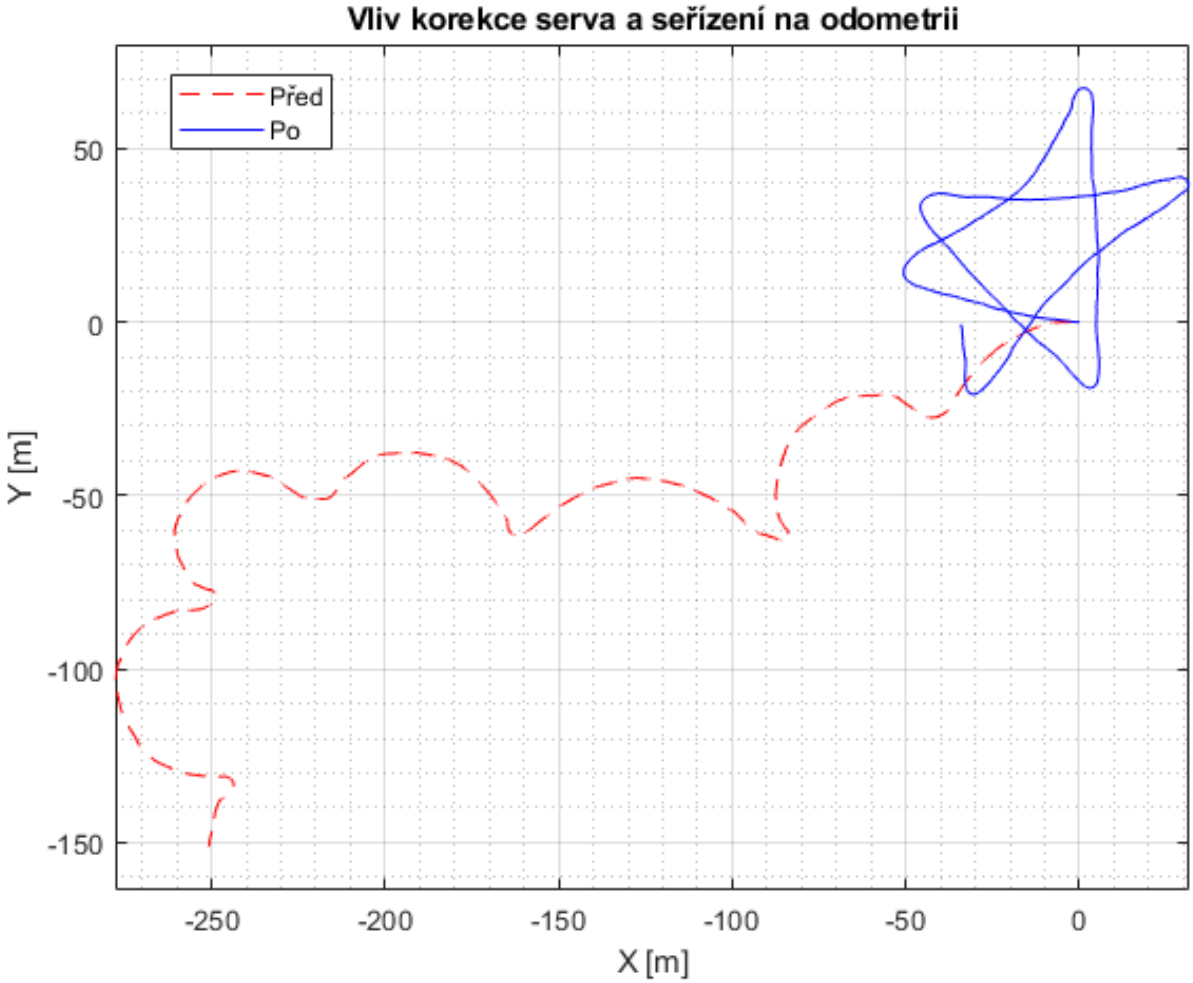


Obrázek 5.7: Závislost úhlu natočení serva na délce řídicího pulzu [12]

Proto bylo nakonec přikročeno k zásahu do *low-level* řízení, který byl proveden jeho tvůrcem a autorem práce [9]. Účelem této úpravy bylo získat možnost nastavit hodnotu PWM pro nulové natočení serva bez zásahu do firmwaru Mbedu. Nově tedy stačí poslat do topicu */servo_change* délku pulzu v sekundách. Použité servo se řídicí pulzy o délce v rozmezí $1 - 2$ ms a hodnota pro nulové natočení kol by tedy měla být přibližně 0.0015 s. Formát zprávy posílané v topicu */servo_change* musí být $std\_msgs :: Float32$.

Obrázek 5.8: Okruh pro testování odometrie [50]

Seřízení odometrie bylo testováno na přibližně trojúhelníkové trase v parku v Lužánkách (obr. 5.8). Tato trasa měří zhruba 280 m a skládá se z asfaltových úseků, dlažebních kostek i sypané cesty. Porovnání odometrie před úpravami a po nich zachycuje obr. 5.9. Jedná se o dva průjezdy trasy ze stejného počátečního bodu. V práci [12] je zmíněno, že na odometrii má velký vliv mimo jiné i povrch, po kterém se robot pohybuje. Uvážíme-li tedy, že na této trase dochází ke střídání povrchů a také její délku, jsou získané výsledky vyhodnoceny jako dobré.



Obrázek 5.9: Porovnání odometrických dat před seřízením a po něm

5.3. Úprava rozlišení obrazu

Robot B2 využívá algoritmus pro rozpoznávání krajů cesty z obrazu. Původně byl pro tyto účely používán obraz s rozlišením 640x480 px. Jak bylo popsáno v kapitole 4.2, byl u robotu B2 během jízdy v autonomním režimu zjištěn problém s nedostatkem výpočetního výkonu a jako jedno z řešení tohoto problému se nabízí snížení rozlišení obrazu.

Při experimentech s obrazem z kamery a jeho vykreslováním v prostředí RViz bylo navíc zjištěno, že při použitém rozlišení 640x480 px dochází ke zpoždění obrazu cca o 2 s. Není sice možné určit, zda k tomuto zpoždění dochází při přenosu obrazu, nebo zda se

může projevit už při jeho zpracování, nicméně snížením rozlišení na čtvrtinu (tj. 160x120 px) byl tento jev téměř eliminován.

I přes to, že se může rozlišení 160x120 px jevit z pohledu člověka jako malé, v tomto případě má hned několik výhod. Čtvrtinové rozlišení znamená, že každý snímek tvoří 16x méně pixelů. Přihlédneme-li navíc k tomu, že byla z důvodu náročnosti zpracování použita vzorkovací frekvence 2 Hz, umožňuje nám tato úprava několikanásobně zrychlit vzorkování, a přes to bude výsledný objem zpracovávaných dat menší. Další výhodou této změny je, že při malém rozlišení dochází do jisté míry k odstranění ostrých přechodů v obrazu, které komplikují jeho zpracování. Výsledná hodnota každého pixelu je totiž v takovémto případě průměrem několika pixelů v CMOS matici snímače.

5.4. Úprava parametrů move_base

Pro navigaci byl v práci [8], která se zabývala lokalizací mobilního robotu B2, zvolen balíček *move_base*. Zmíněná volba byla během realizace této práce kriticky zhodnocena a následně potvrzena. Hlavními důvody jsou poměrně dobrá dokumentace a kompatibilita s mnoha dalšími balíčky.

V konfiguraci *move_base* byl změněn prakticky pouze jeden parametr a to *planner_frequency*. Ten udává frekvenci, s jakou je spouštěn globální plánovač. Vzhledem k tomu, že je použit pro hledání cesty algoritmus prohledávání do šířky (Dijkstra), z jehož podstaty by měl být nalezený plán optimální, byla tato frekvence nastavena na defaultní hodnotu 0.0 Hz. V takovém případě je globální plánovač spuštěn pouze v případě, že je zadán nový cíl, nebo přijde od lokálního plánovače informace o tom, že je cesta neprůjezdná.

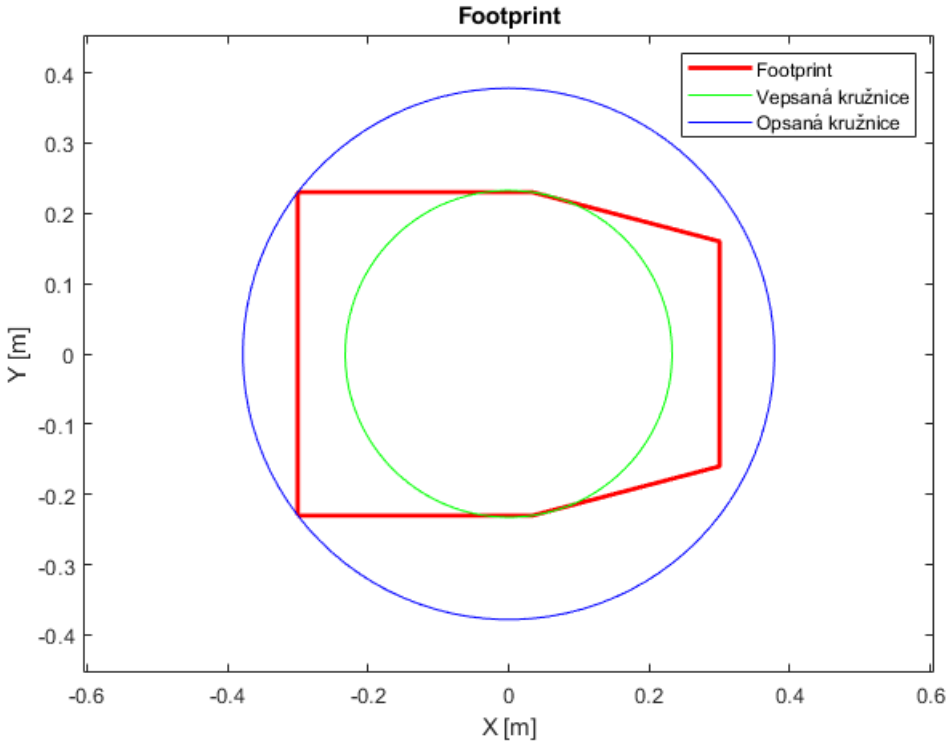
5.4.1. Parametry costmap

Původní nastavení costmap bylo jedním z nejčastějších důvodů selhání autonomní jízdy robotu B2. V případě global costmap docházelo často vlivem nevhodného nastavení senzorů pro zápis překážek a čištění costmapy k tomu, že byl robot po ujetí pár metrů nucen přeplánovat cestu oklikou, neboť vyhodnotil cestu jako neprůjezdnou.

Proto byla zvětšena hodnota parametru *raytrace_range*, aby v případě potřeby došlo k vymazání chybně detekovaných překážek z costmapy dříve a k přeplánování tak nedošlo. Se zvětšením parametru *raytrace_range* je třeba ověřit, zda je zvoleno adekvátní úhlové rozlišení scanneru vzhledem k rozlišení costmapy. Laserový scanner SICK-LMS200 [51] umožňuje pracovat s rozlišením 0.25°, 0.5° a 1°. V ROSu existuje pro práci s těmito scannery balíček *sicktoolbox_wrapper* [52]. V jeho dokumentaci bylo zjištěno, že výchozí nastavení úhlového rozlišení je 1°. Je-li rozlišení lokální costmapy nastaveno na 0.05 m, pak je zbytečné, aby vzdálenost *raytrace_range* byla větší než 2.8 m. Ta přibližně odpovídá maximální vzdálenosti, při které dojde k "prostřelení" všech buněk paprsky scanneru. Zvolená hodnota parametru 1.8 m tedy bez problémů splňuje tuto podmínku a zaručuje správné vyčištění costmapy.

Jak bylo zmíněno, došlo v rámci této práce ke zvětšení robotu. Proto byl podobně jako u parametrů lokálního plánovače upraven *footprint*. Ten zachycuje obr. 5.10.

Dále byly odstraněny některé z *observation sources*. Robot využíval pro zápis do costmap mimo jiné data z topicu */fused/border_array*. Důvod použití těchto dat byl

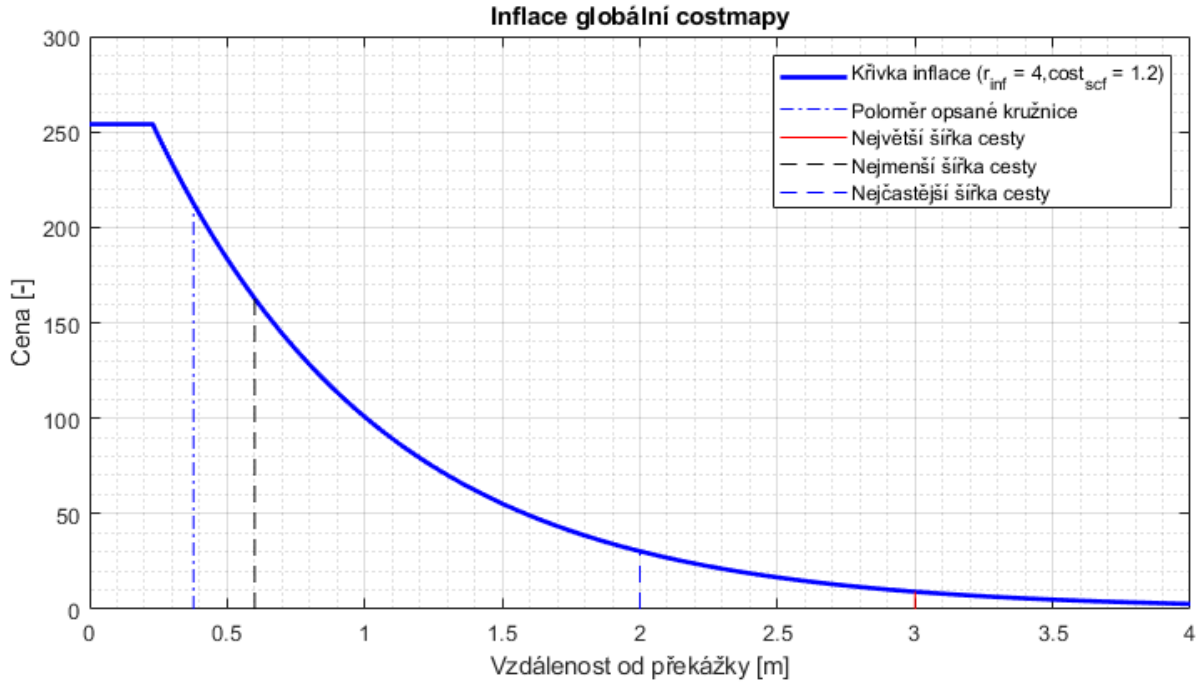


Obrázek 5.10: Footprint robotu B2

takový, že si robot vlastně vytvořil z okrajů cesty "koridor", ve kterém se následně pohyboval [8]. Během měření se však ukázalo, že tato strategie funguje dobře pouze v ideálních podmínkách. Pokud totiž došlo k chybné detekci kraje cesty (např. z důvodu procházejících lidí, stínů na vozovce atd.), robot uvízl. Z tohoto důvodu byl tento zdroj pozorování odstraněn. Zmíněný koridor totiž pouze popisuje okraje cesty, které jsou zahrnuty již ve statické části globální costmapy. Aby byl robot schopen reagovat na překážky, byl ponechán jako zdroj pozorování laserový scanner SICK.

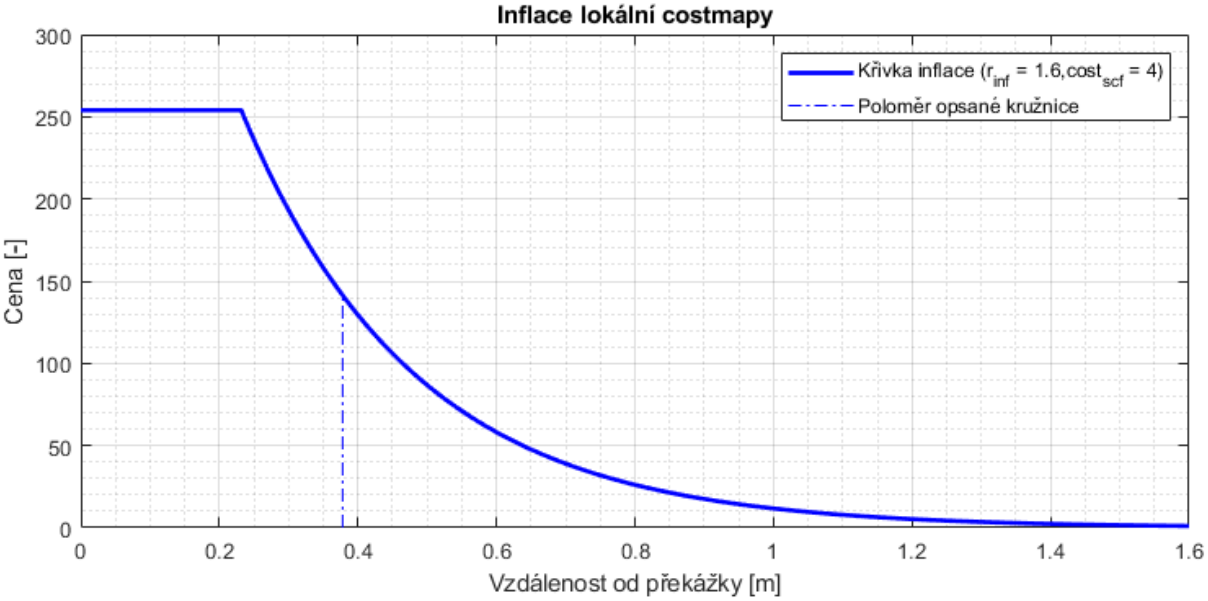
Z důvodu popsaného v kapitole 5.5.3 je žádoucí, aby se globální plán nacházel co nejvíce ve středu cesty. Na to, zda bude splněn tento požadavek, má zásadní vliv nastavení parametrů inflace, resp. parametrů pro výpočet hodnotící funkce. Pro vhodné určení těchto parametrů je nutné znát přibližně šířku cest, po kterých se robot pohybuje, poloměr opsané a poloměr vepsané kružnice footprintu (viz obr. 5.10). Nejprve byla určena hodnota parametru *inflation_radius*. Aby byla nejnižší cena ve středu cesty, měla by hodnota tohoto parametru být větší, než je polovina šířky cesty. Proto byly změřeny šířky cest, na kterých je robot testován. Nejširší cesta je široká 6 m, nejužší 1.2 m a většina cest má šířku kolem 4 m. Proto byl zvolen *inflation_radius* o velikosti 4 m. Poté byl měněn *cost_scaling_factor* tak, aby byl v daném rozsahu zajištěn pokles funkce a zároveň nebyla cena ve středu cesty zbytečně vysoká. Dobrého kompromisu bylo dosaženo při nastavení hodnoty *cost_scaling_factor* na 1.2. Závislost ceny buňky na vzdálenosti od překážky zachycuje graf na obr. 5.11. V něm jsou také pro kontrolu vyznačeny hodnoty zmíněných středů cest a poloměr opsané kružnice. Ten by měl mít cenu ideálně co nejvyšší, aby byla co nejmenší pravděpodobnost kolize. Pro středy cest je zase naopak vhodná cena co nejnižší.

U lokální costmapy byl zvolen opačný postup. Nejprve byla zvolena hodnota *cost_scaling_factor* tak, aby nebyla příliš nízká cena ve vzdálenosti poloměru opsané kružnice.



Obrázek 5.11: Nastavení inflace globální costmapy

Následně byl určen poloměr inflace tak, aby zaokrouhlená cena v této vzdálenosti odpovídala jedné a byl tak zajištěn plynulý přechod mezi volným prostorem a oblastí inflace. Nakonec byla zvolena kombinace parametrů $costmap\_scaling\_factor = 4$ a $inflation\_radius = 1.6$ m. Závislost ceny na vzdálenosti od překážky pro tyto hodnoty zachycuje graf na obr. 5.12. Jak je z grafu patrné, cena buněk ve vzdálenosti, kde ještě teoreticky hrozí při špatném natočení robotu kolize, je přibližně 140. To by mělo zajistit, že robot nebude projíždět v této vzdálenosti, pokud to nebude nezbytně nutné.



Obrázek 5.12: Nastavení inflace lokální costmapy

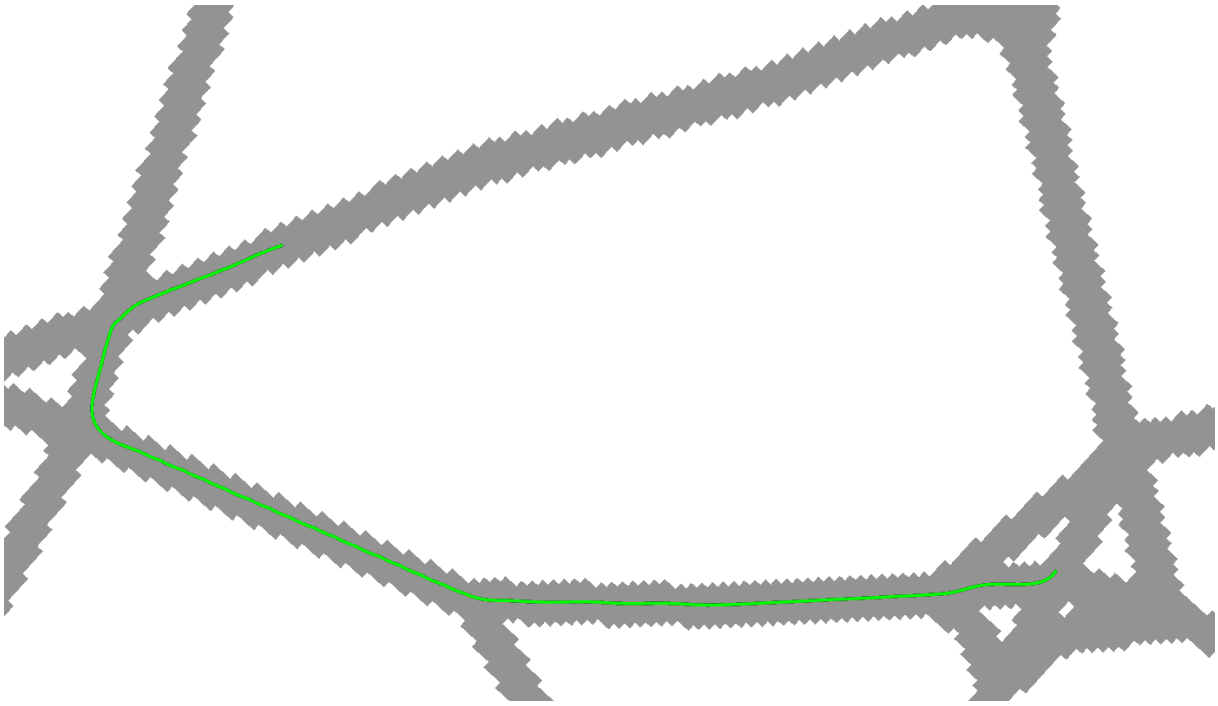
5.4.2. Parametry globálního plánovače

V robotu B2 je použit globální plánovač *global_planner* [26]. Za účelem nalezení vhodných parametrů byla v simulačním prostředí stage [53] otestována řada nastavení. Byly otestovány oba implementované algoritmy a vyzkoušeny různé hodnoty *neutral_cost* a *cost_factor*. Při testování bylo jedním z hlavních kritérií, aby naplánovanou trasu tvořila hladká křivka a to zejména v křižovatkách. To je důležité proto, že je podle tečny této křivky orientován souřadnicový systém *path*, jehož účel je blíže popsaný v kapitole 5.5.4.

Ukázalo se, že při použití algoritmu A* byl nalezený plán více zvlněný než při použití algoritmu Dijkstra a to pravděpodobně vinou použité metriky (viz kapitola 3.3.2). Nejlepších výsledků bylo při testování dosaženo s následujícím nastavením:

```
## Global planner configuration
GlobalPlanner:
  use_dijkstra: true
  neutral_cost: 80
  lethal_cost: 253
  cost_factor: 0.5
```

Ukázku globálního plánu při této konfiguraci zachycuje obr. 5.13, na kterém je vidět, že plán sleduje přibližně střed cesty. To svědčí o dobře zvolených parametrech inflace v globální costmapě. Dále je na obrázku také vidět, že zvlnění plánu je minimální.



Obrázek 5.13: Plán cesty

5.4.3. Parametry lokálního plánovače

V případě nastavení lokálního plánovače bylo provedeno značné množství změn. Tato kapitola je věnována jejich popisu a odůvodnění. Celý konfigurační *.yaml* soubor je pak zařazen v příloze této práce.

Jak bylo popsáno v kapitole 3.3.3, parametry *teb_local_planneru* jsou rozděleny do několika skupin. První z nich je *Robot Configuration Parametres*. Zde byly sníženy hodnoty dovoleného zrychlení za účelem eliminace prokluzu kol a zvýšen minimální poloměr zatáčení tak, aby byla ponechána rezerva pro trimování serva. Dále byl zvětšen parametr footprint pro účely optimalizace lokálního plánu, neboť došlo k rozšíření robotu o nosič pro notebook.

Ve druhé skupině parametrů byla zvětšena tolerance pro dosažení cíle a to nejen tolerance vzdálenosti, ale především tolerance natočení v cílové poloze, neboť jejich hodnota byla vzhledem k prostředí, ve kterém se robot pohybuje, zbytečně malá. To způsobovalo potíže při testování, kdy se robot po příjezdu do cíle nezastavil a stále popojížděl tam a zpět ve snaze přesně trefit zadanou cílovou pozici.

V nastavení *Obstackle Parametres* byla snížena minimální vzdálenost od překážky a zvětšen footprint tak, aby bylo při výpočtech uvažováno zvětšení robotu o zadní desku a zvýšena vzdálenost za robotem, v níž mají být uvažovány překážky během plánování.

V parametrech *Optimization Parametres* byly přiřazeny větší váhy jízdě vpřed, neboť robot často couval i v případech, kdy to evidentně nebylo nutné. Pokud navíc vezmeme v úvahu, že robot není schopen za sebou detekovat překážky, je žádoucí couvání omezit pouze na nezbytně nutné minimum. Dále byla přiřazena vyšší důležitost omezenému poloměru zatáčení.

Poslední skupina, ve které byly provedeny změny, nese název *Parallel Planning in Distinctive Topologies*. Úpravou těchto parametrů bylo dosaženo snížení nároků lokálního plánovače na výpočetní výkon, což mělo zásadní vliv na spolehlivost fungování robotu. Konkrétně zde bylo nastaveno paralelní plánování pouze dvou trajektorií místo původních čtyř, a také zakázána jejich současná optimalizace.

5.5. Změny v softwaru lokalizace

Lokalizace robotu B2 byla vytvořena v rámci diplomové práce [8]. Během tvorby této práce však bylo provedeno několik úprav, přičemž nejzásadnější změnou je doplnění schopnosti lokalizovat se v křižovatce. Detailní popis všech provedených úprav obsahují následující podkapitoly.

Aby dával popis uvedených změn smysl, bude nejprve v tomto odstavci nastíněno, jak původní lokalizace funguje. Podrobné vysvětlení obsahuje již zmíněná práce [8]. Konkrétně je potřeba vysvětlit zejména to, jak se robot lokalizuje ve směru normály střednice cesty (je-li v pravo, vlevo nebo uprostřed). Pokud má robot k dispozici svoji přibližnou polohu (např. z GPS), nalezne nejbližší bod středu cesty k této poloze. K tomuto slouží script *road_from_map.py*. Za zjištění polohy krajních bodů cesty zodpovídají scripty *image_find_path.py* a *sick_find_path.py*. Body získané pomocí těchto scriptů jsou dále fúzovány, k čemuž slouží *fuse_baselink_data.py* a následně je na základě těchto bodů užitím *find_me_on_road.py* zjištěna poloha vůči středu cesty.

5.5.1. Rozpoznávání cesty z obrazu

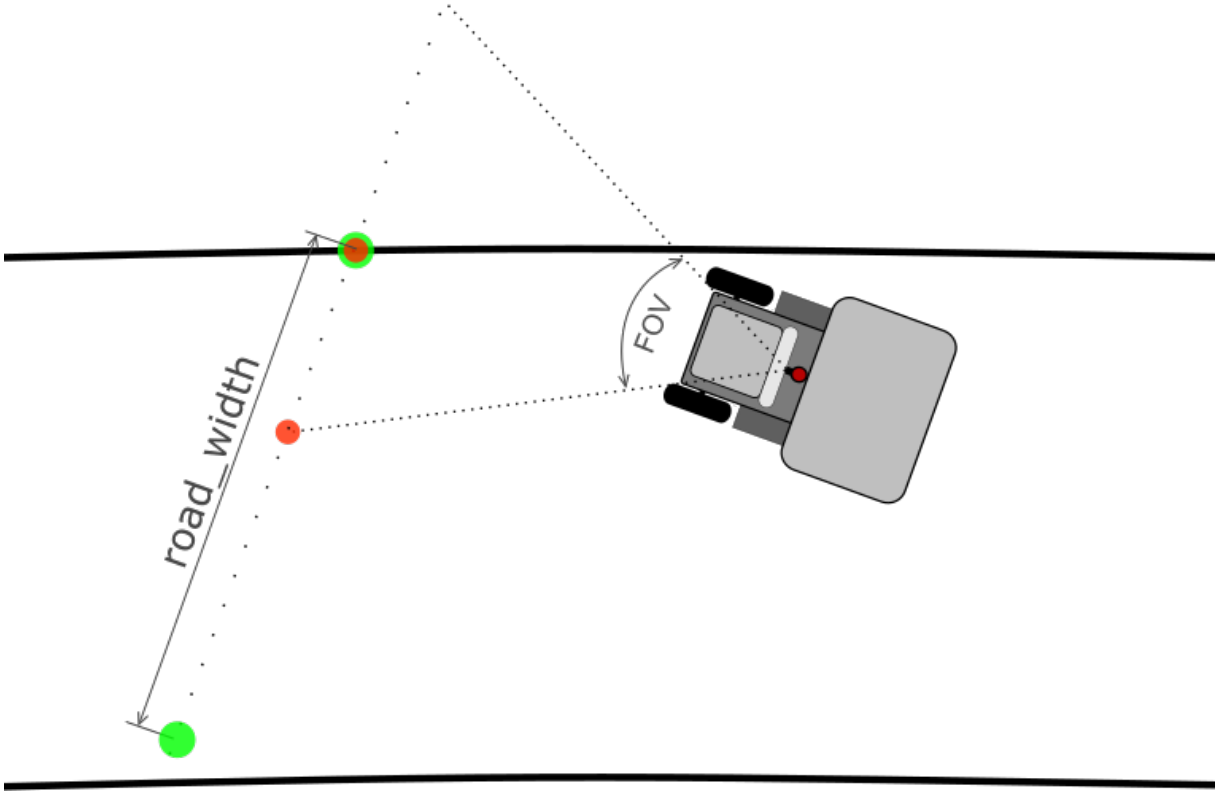
Pro rozpoznávání cesty z obrazu slouží v robotu B2 script *image_find_path.py*, který je dílem autora [8]. Princip fungování tohoto scriptu je následovný: Z obrazu je v určité vzdálenosti od robotu vybrán vodorovný pruh o výšce několika pixelů, které jsou

sečteny. Z tohoto pruhu je vybrána modrá složka obrazu (volba vychází z předpokladu, že asfalt nebo beton mají největší zastoupení této barevné složky). Následně je vytvořen histogram přes šířku pruhu a vypočten práh jako 70 % z maximální intenzity. Hodnoty intenzity nad touto hranicí by měly odpovídat cestě a hodnoty pod ní jejímu okolí. Krajní přechody hodnot v histogramu přes určený práh jsou pak vyhodnoceny jako kraje cesty. Za předpokladu, že se kraj cesty nachází mimo zorné pole kamery, je doplněn na kraj FOV.

Ve scriptu popsaném v předchozím odstavci byly v rámci této diplomové práce provedeny následující změny:

Zavedení parametru šířky cesty

Jako reakce na jev zachycený na obr. 4.2 byl zaveden parametr *road_width*. Za předpokladu, že je detekován pouze jeden kraj cesty a druhý by byl doplněn na kraj FOV kamery, je tento krajní bod cesty umístěn do vzdálenosti *road_width* od nalezeného kraje. Takový stav je zachycen na obr. 5.14 a kromě toho, že lépe odpovídá skutečnosti, hraje navíc klíčovou roli při lokalizaci v křižovatce, která bude popsána v kapitole 5.5.3. Velikost parametru *road_width* byla zvolena 3 m. Tato hodnota přibližně odpovídá šířce vyhodnocovaného pásu ve vyhodnocované vzdálenosti.



Obrázek 5.14: Doplnění kraje cesty

Vizualizace zpracování obrazu

Během jízdy robotu je dobré mít přehled například o tom, jak dobře je schopen detekovat cestu. Pro vykreslení těchto údajů byl v původním scriptu zakomentován kód, který však

vykresloval nalezenou cestu pouze do figure. To ovšem znamená, že se zpracovaný obraz zobrazuje na počítači, kde je script *image_find_path.py* spuštěn. Během zpracování navíc dochází vlivem jedné z použitých funkcí k "poškození" obrazu a k tomu, že je obraz vykreslován jako negativ.

Aby bylo možné sledovat zpracovaný obraz během jízdy na kterémkoli připojeném zařízení, byla provedena úprava scriptu, díky které jsou do původního nepoškozeného obrazu vykresleny žádané údaje a poté je tento obraz streamován do topicu */image_processed*. Výsledek této úpravy zachycuje obr. 5.15. Zelený průsvitný pruh znázorňuje, kde byla nalezena cesta, modře je vykreslen histogram intenzity modré složky obrazu ve vyhodnocovaném pásu, černě zvýrazněný kříž slouží ke kalibraci natočení kamery a vpravo nahoře je vypsáno napětí baterie. V případě, že si robot "myslí", že se nachází v křižovatce, zelený pruh změní barvu na modrou.

Obrázek 5.15: Vizualizace zpracování obrazu

Publikování okrajů v křižovatce

Nachází-li se robot v křižovatce, není vhodné používat pro rozpoznávání krajních bodů SICK a fúzovat data (důvod viz 5.5.3). Proto byl script *image_find_path.py* upraven tak, že v křižovatce publikuje nalezené kraje cesty do topicu */fused/border_pts*. Node lokalizace využívající krajní body tedy bere za všech okolností data z jednoho topicu.

Zavedení příznaku nalezení cesty

Původní script hledal a publikoval kraje cesty neustále a pouze mimo křižovatku byla tato data dále použita. Při průjezdu křižovatkou má však smysl vyhodnocovat data jen za předpokladu, že je vidět alespoň jeden kraj cesty. Pro tento účel byl vytvořen příznak nalezení cesty *priznak_ifp*, který nabývá hodnoty 0, pokud je cesta nalezena, a 1, pokud není vidět ani jeden z okrajů cesty. Pro sdílení hodnoty příznaku ostatním nodům byl vytvořen topic */priznak_ifp*.

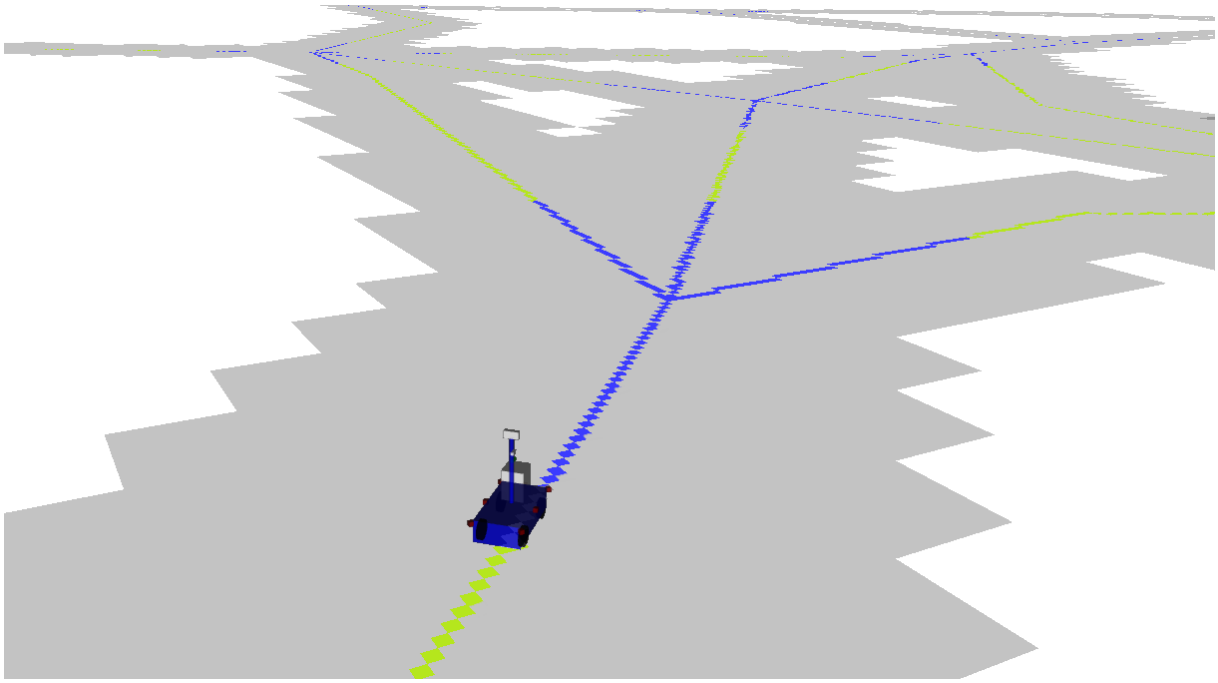
5.5.2. Hledání středu cesty mimo křižovatku

O nalezení středu cesty mimo křižovatku se stará script *road_from_map.py*. Činnost tohoto scriptu je rozdělena do dvou cyklů tzv. "velkého" a "malého". Ve velkém cyklu, jehož výpočet je opakován každých 10 sekund, dochází k ořezání mapy na okolí robotu v určitém rozsahu. V malém cyklu je pak z již redukovaného počtu bodů určován ten nejbližší. V nejbližším bodě je zaveden souřadnicový systém (v terminologii ROSu "frame") *road*. Vzhledem k tomuto souřadnicovému systému je poté dopočítávaná poloha robotu.

V tomto scriptu byla provedena pouze jedna změna a to taková, že byl snížen poloměr ořezané mapy. Původně byla jeho hodnota nastavena na 100 m. To ovšem znamená, že bylo v malém cyklu používáno mnohonásobně více bodů, než je nezbytně nutné. Snížením této vzdálenosti tedy bylo možné ušetřit část výpočetního výkonu. Při volbě nové vzdálenosti bylo uvažováno následovně. Je-li maximální rychlost robotu 0.2 ms^{-1} a délka velkého cyklu 10 s, ujede robot za tento čas maximálně 2 m. Dále je potřeba uvažovat rezervu z důvodu nepřesnosti počáteční polohy určené pomocí GPS, která bývá za nepříznivých podmínek i přes 10 m a to, že mapa nemusí ve všech místech přesně odpovídat skutečným GPS souřadnicím. Nakonec byla zvolena hodnota poloměru pro oříznutí mapy 30 m. Vzhledem k původním 100 m se tedy jedná o zásadní snížení, a zároveň je ponechána dostatečná rezerva pro případné nepřesnosti počáteční lokalizace.

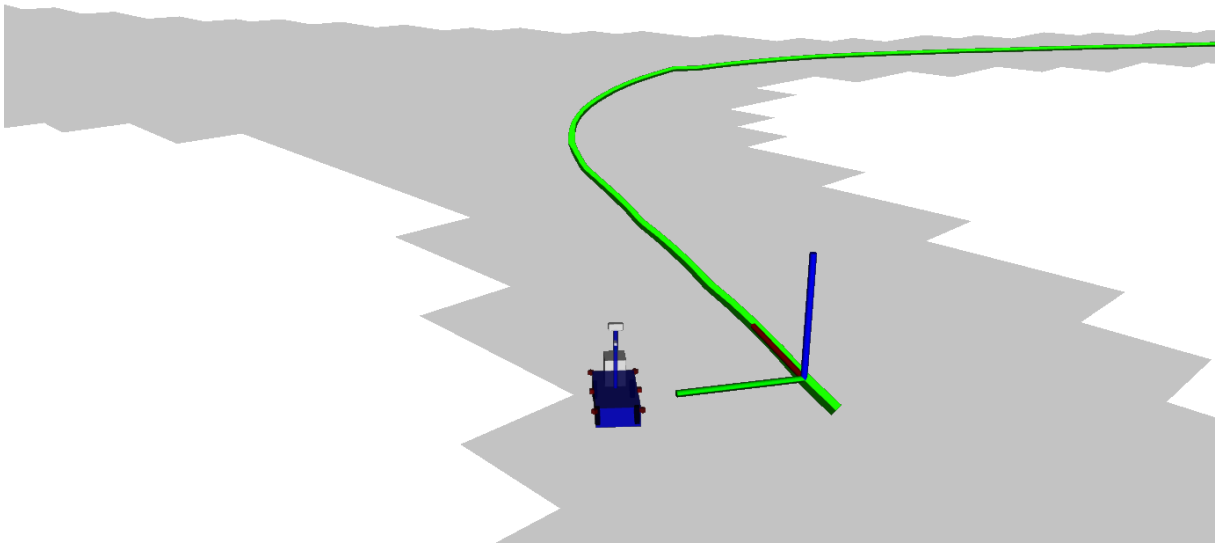
5.5.3. Hledání středu cesty v křižovatce

Script *road_from_map.py* používá k určení nejbližšího středu cesty body uložené při generování mapy do *.txt* (viz obr. 5.16), který mimo souřadnice těchto bodů obsahuje také informace o tom, zda se daný bod nachází v křižovatce. V případě křižovatky však nastává problém, kdy nejbližší bod k robotu nemusí být nutně středem cesty správné odbočky a ve chvíli, kdy dojde k přichycení ke špatné odbočce, je už prakticky nemožné, aby se opětovně správně lokalizoval.



Obrázek 5.16: Mapa s vyznačenými středy cesty vykreslená v RVizu

Tento problém byl vyřešen tak, že je místo zmíněných bodů hledán nejbližší bod z globálnímu plánu, který je při vhodném nastavení parametrů popsaných v kapitole 5.4 přibližně ve středu cesty a neobsahuje žádná rozvětvení. K tomuto účelu byl v rámci této práce vytvořen nový script *road_from_plan.py* inspirovaný *road_from_map.py* a zaveden nový souřadnicový systém *path*. Příklad plánu použitého k určení středu cesty a souřadnicový systém *path* zachycuje obr. 5.17. Protože je orientace *path* určována podle tečny k plánu v daném bodě, je žádoucí, aby plán cesty dával pokud možno hladkou křivku, případně aby obsahoval co nejmenší množství zlomů.



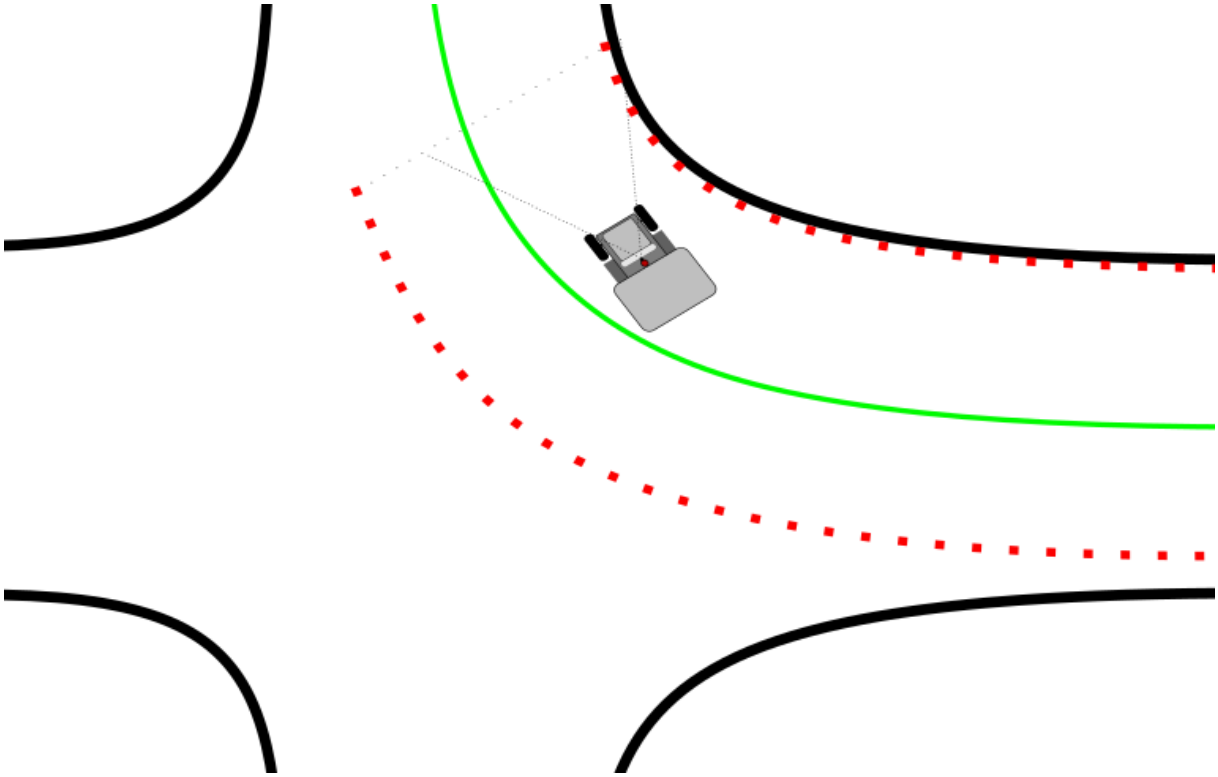
Obrázek 5.17: Určení středů cesty z globálního plánu

5.5.4. Lokalizace vůči středu cesty

Původní script *find_me_on_road.py* rozlišuje pouze dva stavy. V případě, že se robot nachází mimo křižovatku, je určována poloha vzhledem k souřadnicovému systému *road*. Při průjezdu křižovatkou pak původní *find_me_on_road* nepublikuje a robot jede pouze podle odometrie. Toto řešení však často vedlo k problému popsanému v kapitole 4.5.

Upravený script *find_me_on_road.py* rozlišuje více situací. Za předpokladu, že se robot nachází mimo křižovatku, funguje stejně jako původní verze. V případě křižovatky však rozlišuje další dva stavy. Pokud se robot nachází v křižovatce a *priznak_ifp* signalizuje, že je v zorném poli kamery alespoň jeden okraj cesty, probíhá určení polohy stejně, jako je popsáno v práci [8]. Rozdíl je pouze v tom, že je v této situaci místo souřadnicového systému *road* použit souřadnicový systém *path*.

Situaci, kdy robot v křižovatce vidí alespoň jeden okraj cesty a druhý doplní do vzdálenosti *road_width*, zachycuje obr. 5.18. Zelená čára znázorňuje globální plán a červené body pointcloud nalezených okrajů cesty. Robot tedy místo křižovatky projíždí v podstatě "virtuální koridor", v tomto případě ve tvaru zatáčky. Z důvodu výrazně širšího FOV laserového scanneru nejsou jeho data v křižovatce k určení cesty využívána. Fúzování by totiž posunulo doplněný kraj cesty. V případě, že není vidět ani jeden z okrajů, *find_me_on_road* nepublikuje a robot jede podle odometrie, jako tomu bylo původně.

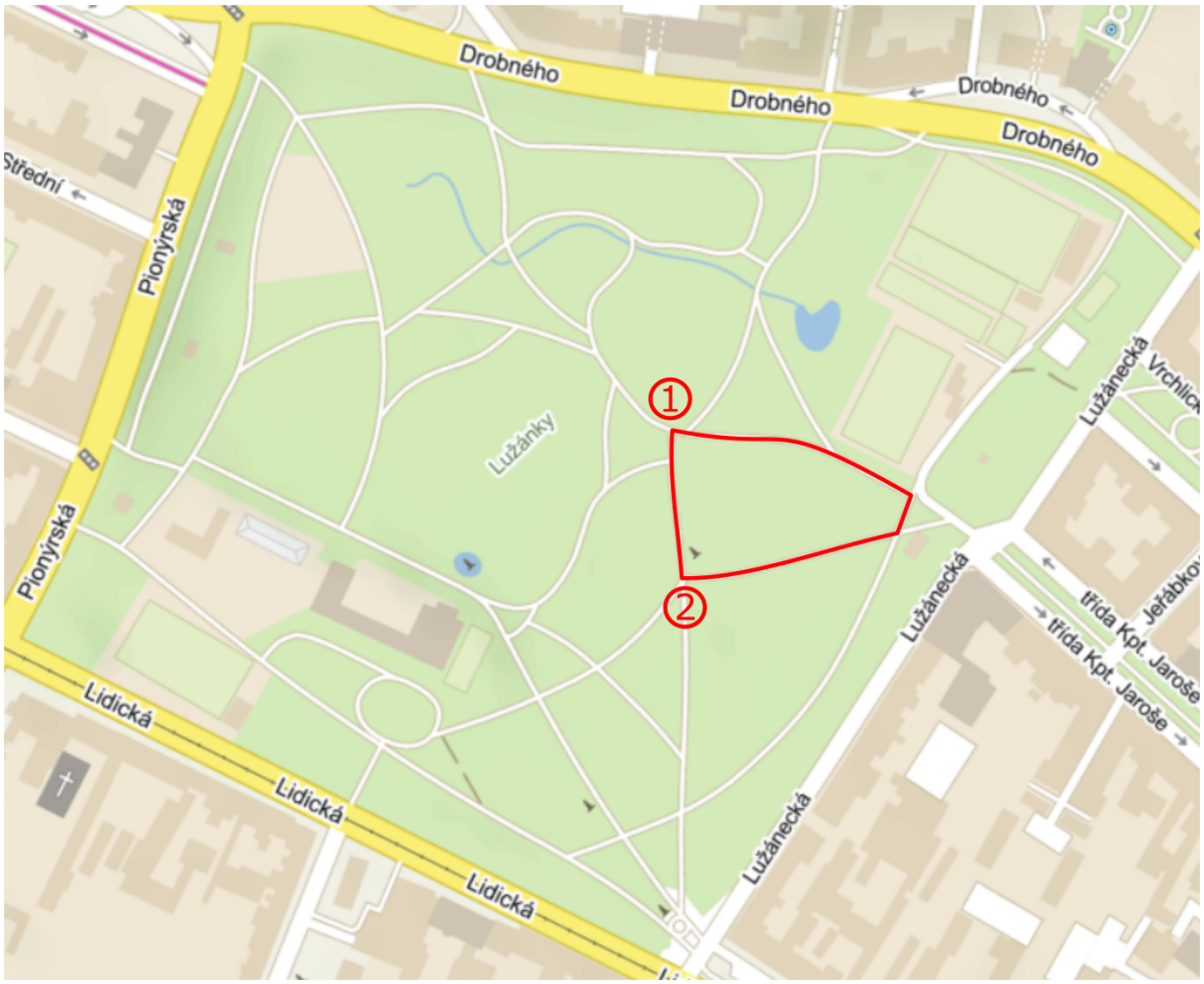


Obrázek 5.18: Průjezd křižovatkou

Tato úprava zabraňuje situacím, kdy robot zatočil před křižovatkou nebo naopak začal zatáčet pozdě a než vyjel z oblasti mimo křižovatku (typicky $2 - 3$ m za ní), odjel zcela mimo cestu. Tímto opatřením se zásadně zvýšila úspěšnost průjezdů křižovatkami.

6. Verifikace a zhodnocení

Pro ověření funkčnosti provedených úprav bylo vymyšleno několik kritérií hodnocení. Samotné testování pak bylo prováděno v parku Lužánky. Mapu parku včetně vyznačeného okruhu, který byl pro tyto účely vybrán, zachycuje obr. 6.1. Hlavním důvodem pro tuto volbu byl fakt, že je tvořen pouze asfaltovými cestami a dlažebními kostkami. Na červených sypaných cestách, které se v parku také vyskytují, totiž nefunguje implementované rozpoznávání obrazu. Délka okruhu je 380 m. Testování probíhalo během několika dní za různých podmínek. Převážně však polojasno až slunečno.



Obrázek 6.1: Mapa testovacího prostředí

6.1. Průjezd křižovatkou

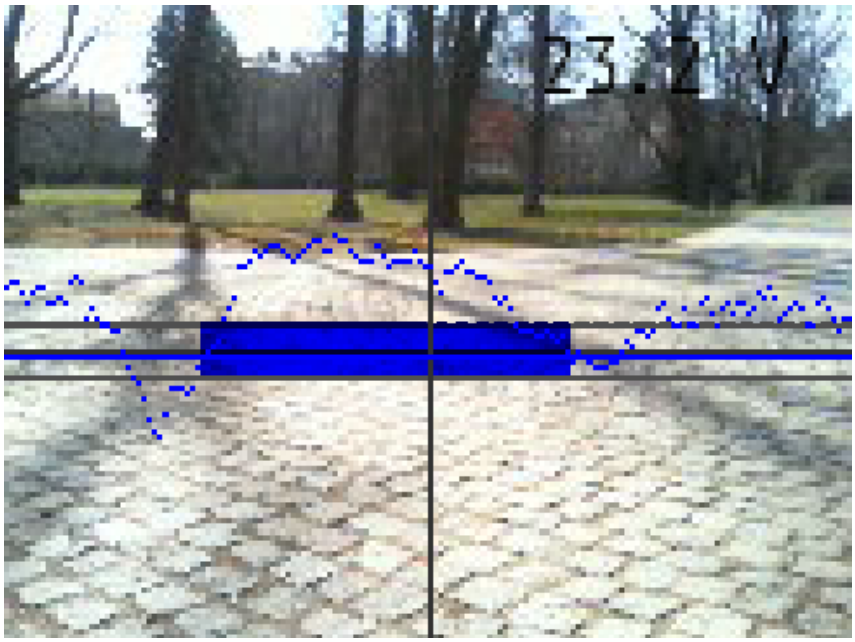
Zvýšení úspěšnosti průjezdů křižovatkou bylo v případě robotu B2 zásadní pro zlepšení schopnosti autonomní jízdy. Proto byla křižovatkám věnována největší pozornost i během závěrečného testování. Konkrétně byly vybrány dvě, které jsou označené v obr. 6.1 jako ”1” a ”2”. Na obou bylo testováno odbočování a na křižovatce ”2” pak i průjezd rovně, a to v obou směrech. Účelem testů bylo zejména posoudit schopnost robotu vyrovnat se s nesprávně určenou počáteční polohou (viz kap. 4.5).

Protože údaje z lokalizace robotu B2 popisují pouze to, kde si robot "myslí", že je, a z principu neodpovídají skutečné poloze, nebylo prakticky možné kvantifikovat kvalitu konkrétního pokusu a pouze byly rozlišovány stavy průjezd úspěšný a průjezd neúspěšný. Tato úspěšnost se pohybovala před provedením úprav kolem 10 − 15 %. Výsledky testování po úpravě zachycuje tab. 6.1 a podrobnosti k vybraným křižovatkám budou rozepsány v následujících podkapitolách.

	Počet pokusů	**Úspěšné**	**Spolehlivost**
Křižovatka "1"	19	15	79 %
Křižovatka "2"	20	20	100 %
Křižovatka "2" rovně	20	17	85 %

Tabulka 6.1: Úspěšnost průjezdů křižovatkou

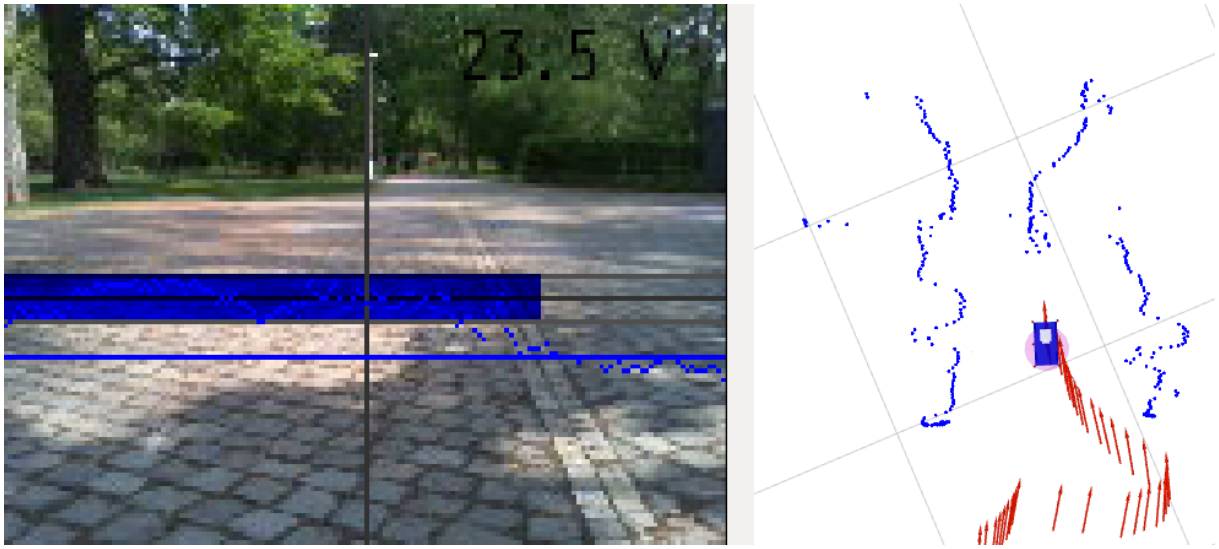
Jak je z tabulky patrné, doplnění lokalizace o "křižovatkovou logiku" zásadním způsobem zlepšilo schopnost projet křižovatkou. Nejčastější příčinou selhání byla přítomnost stínů na vozovce například od stromů. Důležitou roli zde však měla i jejich orientace vzhledem k vozovce. V případě stínů kolmých k cestě zásadní problémy nevznikaly. Situace, kdy dva sousední stíny a plán cesty svíraly ostřejší úhel (viz obr. 6.2) vedla naopak velmi často k selhání. Za těchto okolností se robot obvykle zaměřil na světlý pás a pokračoval jeho směrem.



Obrázek 6.2: Přichycení k falešné cestě

Ve chvíli, kdy robot takto dojede do místa, kde nemá ve vyhodnocovaném pásu ani kousek cesty, je šance na její návrat prakticky nulová. Algorimus vyhodnocuje intenzitu modré složky obrazu relativně v rámci použitého pásu a i na trávě pak vybírá světlejší místa, která považuje za cestu.

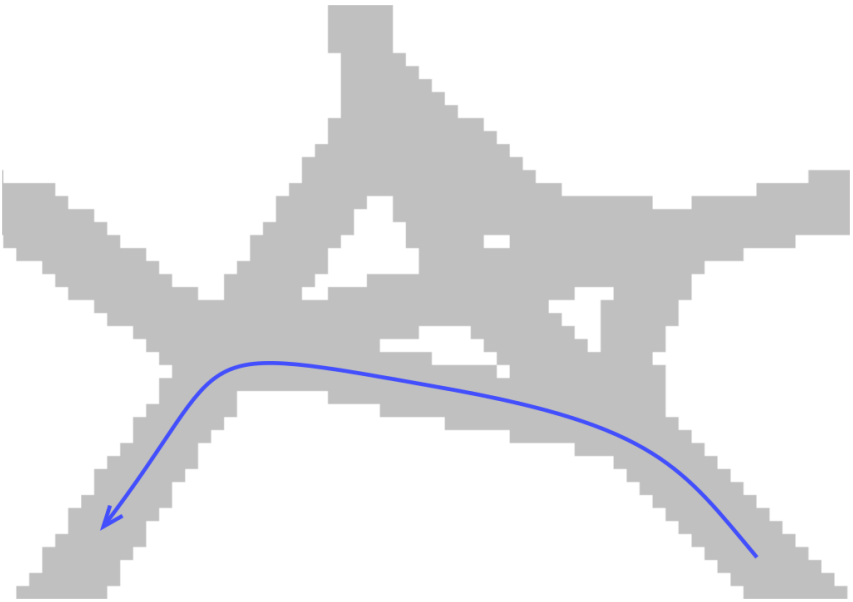
Pokud stíny netvoří rovné linky (typicky stín od koruny stromu), problém nebývá tak zásadní. Body point cloudu v takové situaci sice vypadají jako náhodně rozeseté po ploše v okolí robotu, ale při počítání průměrných hodnot využívaných při lokalizaci obvykle nezpůsobí takovou chybu. Tuto situaci zachycuje obr. 6.3.



Obrázek 6.3: Chybná detekce krajů

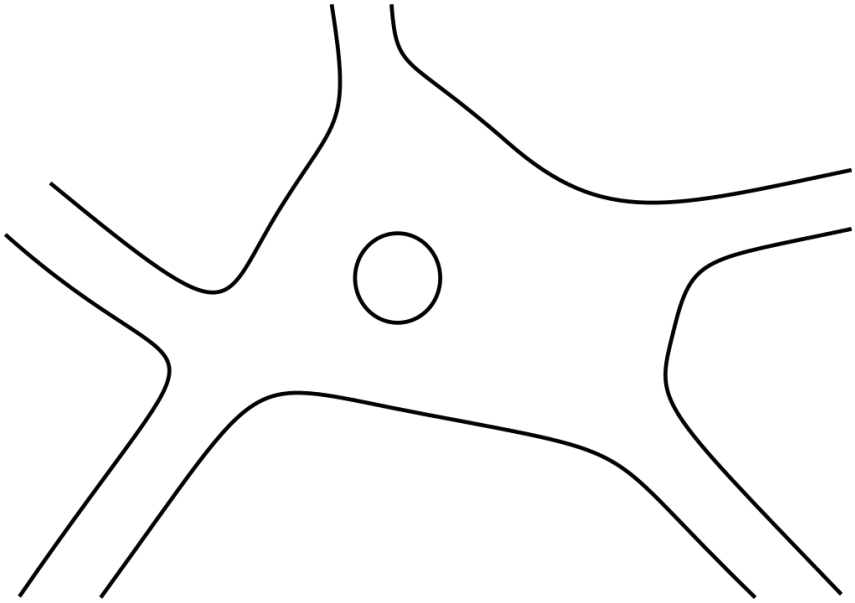
Křižovatka ”1”

Na této křižovatce bylo testováno odbočování vlevo. Důvodem výběru konkrétně této křižovatky byla mimo jiné skutečnost, že její podoba neodpovídá mapě, podle které se robot orientuje. Vzhledem k tomu, že při používání bitmapy vygenerované z Open Street Maps nemusí být takový jev ojedinělý, bylo učiněno rozhodnutí otestovat schopnost navigace vyrovnat se s popisovaným problémem. Příslušný výřez mapy, kterou má robot k dispozici, s vyznačeným plánem cesty zachycuje obr. 6.4.



Obrázek 6.4: Křižovatka ”1” na mapě použité v robotu

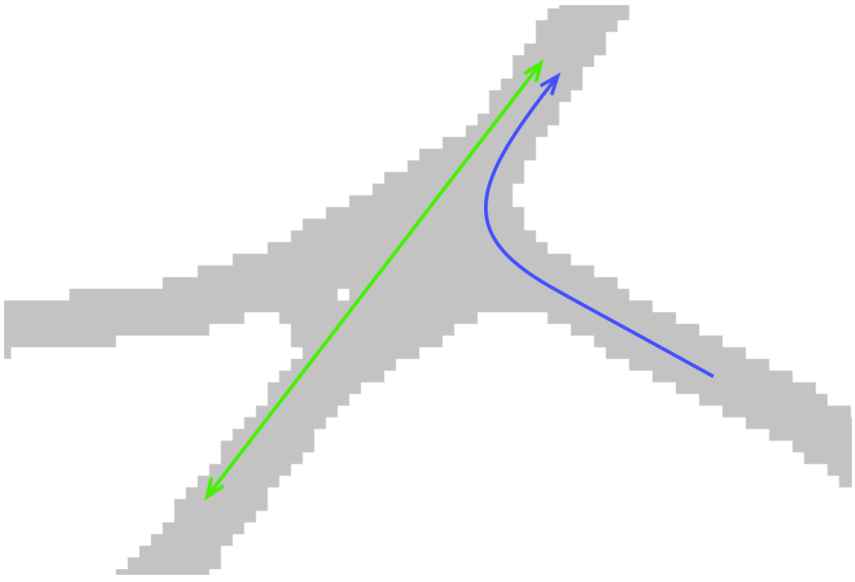
Ve skutečnosti se však přibližně ve středu křižovatky nachází strom a její tvar mnohem lépe vystihuje obr. 6.5. I přes to však byla úspěšnost na této křižovatce nad očekávání dobrá.



Obrázek 6.5: Skutečná podoba křižovatky ”1”

Křižovatka ”2”

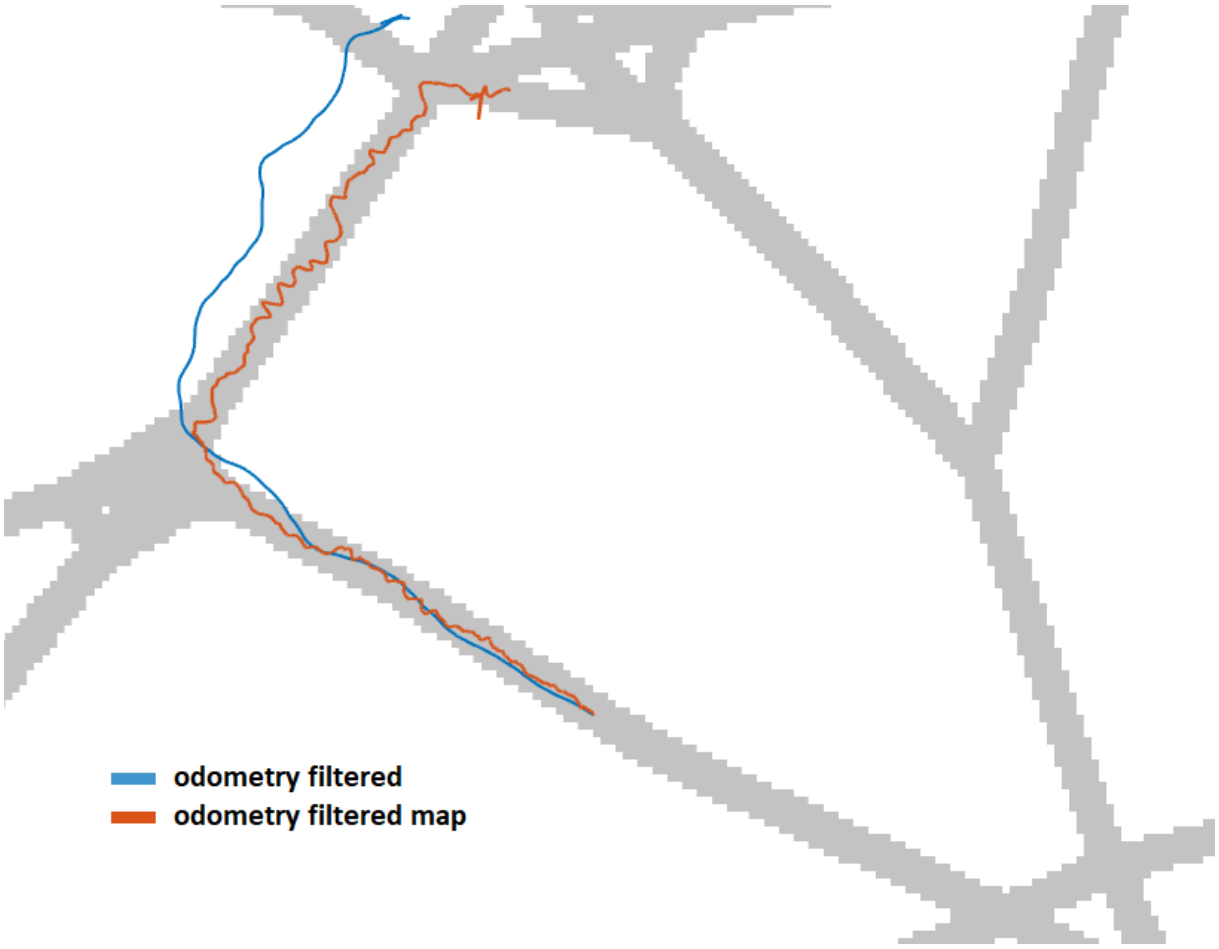
Tvar křižovatky ”2” je zachycen na obr. 6.6, kde je opět vyznačen také směr jízdy. Zde mapa dobře odpovídá skutečnosti a také jde o typičtější tvar křižovatky. Zejména díky prvnímu z uvedených faktorů je úspěšnost v této křižovatce vyšší. Přesto byl právě na křižovatce ”2” nejvíce pozorován vliv šikmých stínů. Během jednoho ze dnů, kdy bylo prováděno měření, byla totiž úspěšnost pouze 50 %. Ve zbylých dnech, kdy měření probíhalo v dřívějších hodinách, a poloha stínů byla odlišná, se tato křižovatka jevila jako bezproblémová i za slunečného počasí.



Obrázek 6.6: Křižovatka ”2”

6.2. Jízda po cestě

Aby bylo otestováno i chování na rovné cestě, bylo s robotem uskutečněno několik delších jízd, během nichž ujel vždy $100-150$ m v autonomním režimu. Příklad jedné z těchto jízd zachycuje obr. 6.7. V něm jsou do mapy vykreslena data z topicu */odometry_filtered_map*, která zachycují to, jak se robot během jízdy lokalizoval. Na těchto datech je však na první pohled vidět, že se sice nacházejí na cestě, ale zcela jistě neodpovídají přesně skutečnosti, neboť robot není schopen jízdy do stran. Proto byla k porovnání vykreslena i data z topicu */odometry_filtered*. Ta jsou získávána pouze pomocí prostředků relativní lokalizace (viz práce [8]) a podléhají tedy driftu, nicméně pro krátké úseky lépe vystihují skutečnou dráhu robotu.



Obrázek 6.7: Kličkování během jízdy

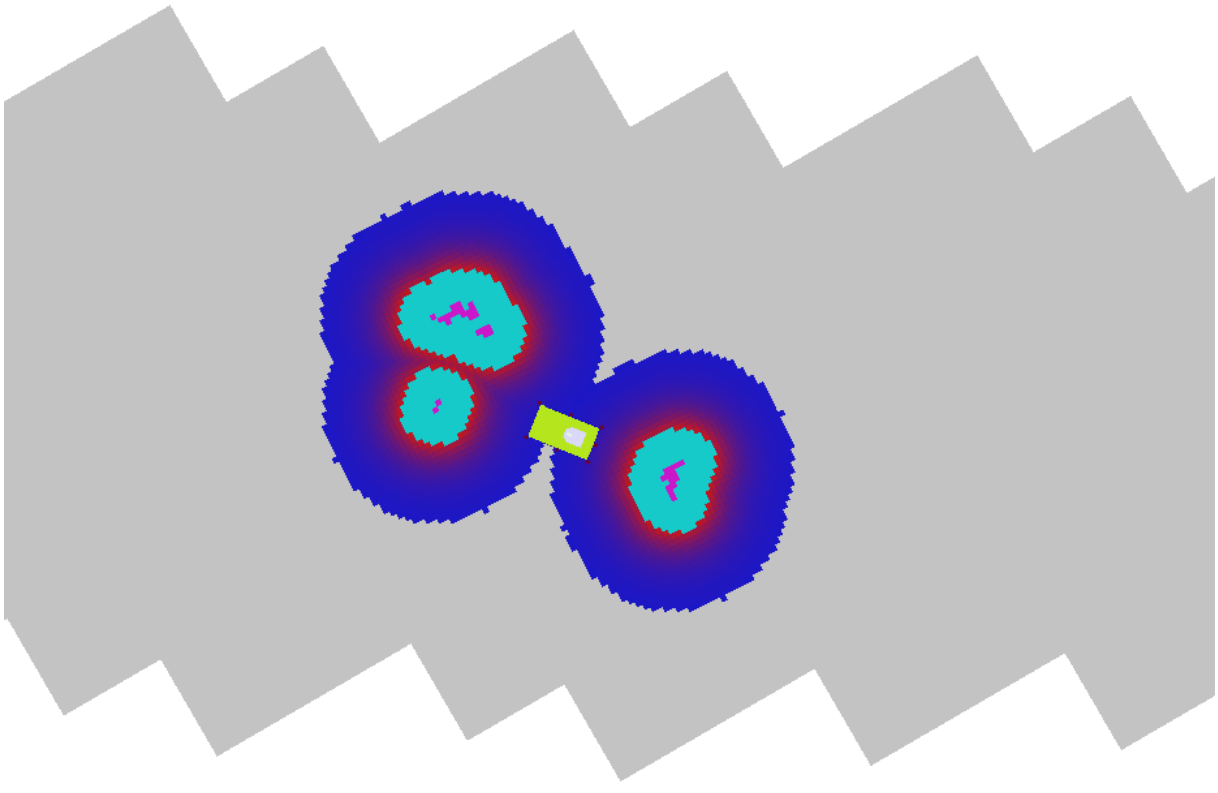
Hlavní příčinou výskytu skokových změn polohy v lokalizaci je pravděpodobně opět problém s rozpoznáváním cesty. Tento jev pak navíc negativně ovlivňuje navigaci, která v reakci na kolísání polohy a snaze držet se plánu cesty nutí robota kličkovat, čímž celou situaci zhoršuje. Skoky v lokalizaci jsou také do jisté míry způsobeny tím, že jedním ze zdrojů pro určování polohy je GPS. Ta ze své podstaty neurčuje spojitý průběh polohy. Dalším faktorem, který může mít negativní vliv na stabilitu jízdy, je zpoždění, které vzniká během zpracování dat. Při limitu rychlosti 0.2 ms^{-1} by se sice standardní zpoždění nemělo projevit, avšak byly pozorovány situace, kdy pravděpodobně došlo k chvilkovému přetížení palubního počítače, což mělo za následek opožděnou reakci robotu.

6.3. Chování v dynamickém prostředí

U schopnosti reagovat na překážky došlo u robotu B2 oproti původnímu stavu ke značnému zlepšení. Od provedení změn nebylo nutné použít během jízdy v autonomním režimu k zabránění kolize ani jednou nouzové STOP tlačítko. Hlavní zásluhu na tomto zlepšení má snížení vytíženosti palubního počítače a přenastavení parametrů navigace. Testování však ukázalo, že i přesto nebyly vyřešeny zdaleka všechny problémy.

Prvním z nich je, že jsou překážky z principu fungování algoritmů identifikovány jako kraj cesty, a to v tomto případě nejen u rozpoznávání z obrazu, ale i při vyhodnocování dat ze SICKu. To obvykle nevadí u pohybujících se překážek, které se nenachází v dosahu senzorů příliš dlouho. Pokud se ale na cestě před robotem vyskytují delší dobu objekty, které nejsou zahrnuty v mapě, robot se jim sice vyhne, ale zároveň nezřídka dojde ke zmatení lokalizace a následnému selhání navigace.

Druhý problém souvisí s již zmíněnou absencí senzorů v zadní části robotu. Ta totiž vede kromě toho, že za sebou robot nemůže detekovat překážky, také k tomu, že v tomto prostoru nemůže čistit costmapu. Příklad situace, která nastala z tohoto důvodu, zachycuje obr. 6.8. Prostor za robotem (na obrázku vlevo) je ve chvíli, kdy robot (obdélník uprostřed) detekuje překážku před sebou, čistý. Protože se ale vlivem jízdy robotu dostali dříve detekované překážky mimo zorné pole laserového scanneru, zůstaly zapsány v costmapě a robot tak uvízl do doby, než byla přední překážka odstraněna.



Obrázek 6.8: Uvíznutí robotu

6.4. Inicializace

Poslední z kritérií hodnocení byla spolehlivost inicializace při spuštění autonomního režimu. V tomto ohledu bylo vyhodnoceno z 80 pokusů 13 jako neúspěšných, což z procentuálního hlediska činí úspěšnost 83.75 %. To není příliš mnoho, nicméně je potřeba od sebe rozlišit minimálně dva typy selhání. První typ se projevoval tak, že robot sice určil svoji polohu, vytvořil plán a začal posílat data do *low-level* řízení, ale přes to se nerozjel. V takovém případě bylo nutné robota úplně vypnout a zapnout.

Druhý typ selhání souvisel s tím, že byly testovány především křižovatky. Pokud totiž robot dostal chybné GPS souřadnice, které již spadaly do oblasti s příznakem křižovatky, nebyl schopen se přichytit ke středu cesty. K tomu totiž v křižovatce používá globální plán, který není možné vytvořit z místa mimo cestu. Oblasti mimo cestu mají totiž v costmapě hodnotu ceny *lethal_cost* a výpočet tedy selže. V situaci, kdy takto chybně určená poloha z GPS spadla do území cesty, se robot bez problémů po zadání cíle rozjel. Tento případ se týkal 6 ze 13 selhání. A v případě spouštění navigace dál od křižovatky by úspěšnost inicializace byla pravděpodobně o něco lepší. Velká chyba určení počáteční polohy (i přes 10 m) by s nejvyšší pravděpodobností stejně způsobila problémy později během jízdy.

7. Návrhy pro další práci

Během této práce byl učiněn viditelný pokrok v autonomním řízení robotu B2. Jedná se ovšem o natolik komplexní a složitou úlohu, že stále zůstává značný prostor pro zlepšení. Na základě poznatků získaných během tvorby této práce je doporučeno zvážit následující návrhy pro zlepšení:

- Zřejmě největší slabinou robotu B2 je v současné chvíli rozpoznávání cesty, které je klíčové pro úspěch navigace. V rámci této práce byla testována připravená neuronová síť. Ta však hledala pouze čtyři krajní body cesty, nefungovala příliš spolehlivě a zpracování jednoho obrázku o rozlišení 640x480 px trvalo přibližně 500 ms na procesoru Intel Core i7. Vzhledem k tomu, že je procesor v robotu B2 v současnosti už prakticky plně vytížen, je doporučeno prozkoumat možnosti ohledně implementace CNN (Convolution Neural Network) na FPGA. Té se věnuje například článek [54], kde je stručně popsána struktura použité sítě a srovnání výsledků z CPU, GPU a FPGA. Ve stručnosti se dá říci, že autoři dosáhli s FPGA téměř stejné úspěšnosti jako s GPU, ovšem s řádově menší spotřebou energie a podstatně levnějším zařízením.

- Dále je doporučeno zvážit modernizaci *low-level* části robotu. Konkrétně přidat senzory na zadní stranu a zkontrolovat kabeláž. Během posledního měření bylo zjištěno, že při pohnutí s kabelem od baterie někdy vypadne napájení. Příčina je pravděpodobně v pouzdře pro pojistky. Také by bylo vhodné přidat do *low-level* řízení timeout, po kterém dojde k zastavení robotu, nepřijde-li nová instrukce z řízení. Současné řešení totiž nemá implementovanou žádnou pojistku pro případ selhání nebo neplánovaného ukončení *high-level* řízení a robot v takové situaci pokračuje v jízdě podle poslední příchozí instrukce.

- Robot B2 má v současné době nainstalovanou distribuci ROS Indigo Igloo, které již skončila podpora. Nově je doporučováno použít distribuci Melodic Morenia s prodlouženou podporou do roku 2023 [55].

8. Závěr

Úvodní kapitoly této práce obsahují stručný popis robotu B2 a popis některých knihoven frameworku ROS souvisejících s navigací. Konkrétně se jedná o popis různých typů reprezentace mapy použitelných pro účely navigace, popis knihovny *move_base* a popis vybraných plánovačů.

Protože robot B2 není dílem této práce, byl nejprve zkoumán jeho výchozí stav. To vedlo k odhalení některých nedostatků, z nichž nejzásadnější bylo nadměrné vytížení procesoru palubního počítače a neschopnost lokalizovat se v křižovatce. Další zjištěné nedostatky byly méně významného charakteru a týkaly se konstrukce, nastavení plánovačů a uživatelské přívětivosti.

Za účelem zlepšení navigace byly v rámci této práce zkalibrovány některé senzory robotu a byly provedeny úpravy hardwaru i softwaru. Jmenovitě jsou to například snížení rozlišení obrazu z kamery a omezení některých parametrů lokálního plánovače tak, aby kleslo vytížení procesoru na únosnou hodnotu, doplnění schopnosti lokalizace při průjezdu křižovatkou a úprava parametrů navigace.

Po úpravách byl robot za účelem ověření funkčnosti provedených opatření opět testován. Nedostatků odhalených během testování je několik. Robot má tendenci při jízdě po rovině kličkovat. Jednou z příčin je pravděpodobně nedokonalost použitých metod pro rozpoznávání cesty, která je navíc dále zhoršována výskytem stínů a nerovnostmi vozovky. U širších cest se zde projevuje i fakt, že zorné pole kamery neobsáhne celou šíři cesty. Dále byl i přes značné snížení nároků na výkon pozorován jev, kdy došlo k chvilkovému přetížení počítače v robotu, čímž vzniklo nestandardní zpoždění ve zpracování dat. To v krajních případech vede k selhání navigace.

I přes přetrvávající nedokonalosti bylo ohledně schopnosti autonomní jízdy zaznamenáno zásadní zlepšení, a to zejména v oblasti průjezdů křižovatkou, kde se úspěšnost zvýšila z původních přibližně 15 % v průměru na 88 % (podle konkrétní křižovatky). Dále bylo zaznamenáno zásadní zlepšení ve schopnosti reagovat na dynamické prostředí. Od provedení změn již nebylo při jízdě v autonomním režimu zapotřebí zásahu zvenčí z důvodu hrozící kolize.

Cílem práce bylo zlepšit navigační schopnosti mobilního robotu B2. Na základě výsledků testování lze říci, že spolehlivost navigace stále není zcela stoprocentní a je zde pořád prostor pro zdokonalení. Zlepšení, kterého bylo dosaženo oproti původnímu stavu, je však výrazné a cíl práce byl tedy splněn.

Literatura

1. *FSD chip* [online] [cit. 2019-05-11]. Dostupné z: https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip.
2. *ROS* [online] [cit. 2019-05-12]. Dostupné z: http://www.ros.org/.
3. *Willow Garage* [online] [cit. 2019-05-12]. Dostupné z: http://www.willowgarage.com/.
4. *ROS Industrial* [online] [cit. 2019-05-12]. Dostupné z: https://rosindustrial.org/.
5. *ROS 2* [online] [cit. 2019-05-12]. Dostupné z: https://fkromer.github.io/awesome-ros2/.
6. *ROS wiki* [online] [cit. 2019-05-17]. Dostupné z: http://wiki.ros.org.
7. TOMÁŠ, Petr. *Návrh a realizace senzorického systému pro mobilní robot s využitím frameworku ROS*. Brno, 2014. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství.
8. KORYTÁR, Lukáš. *Realizace lokalizačního systému pro mobilní robot B2*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství.
9. MEINDL, Jan. *Návrh a realizace vestavěného systému řízení mobilního robotu Bender II*. Brno, 2015. bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství.
10. *Indigo* [online] [cit. 2019-04-19]. Dostupné z: http://wiki.ros.org/indigo.
11. VENC, Luboš. *Návrh a realizace navigačního systému pro mobilní robot Bender II*. Brno, 2018. bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství.
12. PORTEŠ, Petr. *Návrh a realizace odometrických snímačů pro mobilní robot s Ackermannovým řízením.* Brno, 2017. diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství.
13. *Simple DirectMedia Layer* [online] [cit. 2019-04-19]. Dostupné z: http://www.libsdl.org/.
14. *Map server* [online] [cit. 2019-04-19]. Dostupné z: http://wiki.ros.org/map_server.
15. *Navigation* [online] [cit. 2019-04-19]. Dostupné z: http://wiki.ros.org/navigation.
16. *Open Street Maps* [online] [cit. 2019-04-19]. Dostupné z: https://www.openstreetmap.org/.
17. *OSM cartography* [online] [cit. 2019-04-19]. Dostupné z: http://wiki.ros.org/osm_cartography.
18. *Marker Array* [online] [cit. 2019-04-19]. Dostupné z: http://docs.ros.org/api/visualization_msgs/html/msg/MarkerArray.html.
19. *Route Network* [online] [cit. 2019-04-19]. Dostupné z: http://wiki.ros.org/route_network.

20. *OSM planner* [online] [cit. 2019-04-19]. Dostupné z: `https://github.com/MichalDobis/osm_planner/wiki`.
21. DOBEŠ, Jan. *Plánování cesty v mapě OpenStreet*. Brno, 2014. bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
22. *Large Maps Framework* [online] [cit. 2019-04-19]. Dostupné z: `http://wiki.ros.org/Large\%20Maps\%20Framework`.
23. *Move Base* [online] [cit. 2019-04-20]. Dostupné z: `http://wiki.ros.org/move_base`.
24. *Carrot planner* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/carrot_planner`.
25. *Navfn* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/navfn`.
26. *Global planner* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/global_planner`.
27. *Base local planner* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/base_local_planner`.
28. *DWA planner* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/dwa_local_planner`.
29. *Voronoi planner* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/voronoi_planner`.
30. *Eband planner* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/eband_local_planner`.
31. *Teb local planner* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/teb_local_planner`.
32. *Move base flex* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/move_base_flex`.
33. *Costmap 2D* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/costmap_2d`.
34. *Voxel grid* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/voxel_grid`.
35. *Obstacle Avoidance and Robot Footprint Model* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/teb_local_planner/Tutorials/Obstacle\%20Avoidance\%20and\%20Robot\%20Footprint\%20Model`.
36. *Static map* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/costmap_2d/hydro/staticmap`.
37. *Obstackles* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/costmap_2d/hydro/obstacles`.
38. *Pcl ros* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/pcl_ros`.
39. *Scan tools* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/scan_tools`.
40. *Inflation* [online] [cit. 2019-04-22]. Dostupné z: `http://wiki.ros.org/costmap_2d/hydro/inflation`.

41. *The taxicab metric* [online] [cit. 2019-04-22]. Dostupné z: http://www.oxfordmathcenter.com/drupal7/node/386.
42. *ROS planning* [online] [cit. 2019-04-22]. Dostupné z: https://github.com/ros-planning/navigation/blob/indigo-devel/navfn/include/navfn/navfn.h.
43. ZHENG, Kaiyu. ROS Navigation Tuning Guide [online]. 2017, s. 19 [cit. 2019-04-22]. Dostupné z: https://www.researchgate.net/publication/318011822_ROS_Navigation_Tuning_Guide.
44. *Set up and test Optimization* [online] [cit. 2019-04-22]. Dostupné z: http://wiki.ros.org/teb_local_planner/Tutorials/Setup\%20and\%20test\%20Optimization.
45. *Ackerman msg* [online] [cit. 2019-04-22]. Dostupné z: http://wiki.ros.org/ackermann_msgs.
46. *Costmap converter* [online] [cit. 2019-04-22]. Dostupné z: http://wiki.ros.org/costmap_converter.
47. *Htop* [online] [cit. 2019-04-29]. Dostupné z: https://hisham.hm/htop/.
48. *RViz* [online] [cit. 2019-04-06]. Dostupné z: http://wiki.ros.org/rviz.
49. *MPU-9150* [online] [cit. 2019-04-24]. Dostupné z: https://www.invensense.com/wp-content/uploads/2015/02/MPU-9150-Datasheet.pdf.
50. *Mapy cz* [online] [cit. 2019-04-30]. Dostupné z: https://mapy.cz.
51. *SICK-LMS200* [online] [cit. 2019-05-03]. Dostupné z: https://www.sick.com/cz/cs/reseni-mericich-a-detekcnich-aplikaci/2d-lidar-senzory/lms2xx/lms200-30106/p/p109843.
52. *Sick toolbox wrapper* [online] [cit. 2019-05-03]. Dostupné z: http://wiki.ros.org/sicktoolbox_wrapper.
53. *Stage* [online] [cit. 2019-04-27]. Dostupné z: https://codedocs.xyz/CodeFinder2/Stage/index.html.
54. ZHAI, Sheping; QIU, Cheng; YANG, Yuanyuan; LI, Jing; CUI, Yiming. *Design of Convolutional Neural Network Based on FPGA*. 2019. Č. 6. ISSN 1742-6588. Dostupné z DOI: 10.1088/1742-6596/1168/6/062016.
55. *ROS Melodic Morenia* [online] [cit. 2019-05-04]. Dostupné z: http://wiki.ros.org/melodic.

9. Seznam použitých zkratek a symbolů

FSD	Full Self-Driving - počítač vyvinutý pro účely autonomního řízení
ROS	Robot Operating System - framework pro tvorbu robotických aplikací
BSD	Berkeley Software Distribution - licence pro svobodný software
IMU	Inertial Measurement Unit - inerciální měřící jednotka
GPS	Global Positioning System - globální polohový systém
SDL	Simple DirectMedia Layer - knihovna v jazyce C pro přístup k hardwaru
SQL	Structured Query Language - strukturovaný dotazovací jazyk používaný pro práci s databázemi
LaMa	Large Maps - framework pro navigaci v rozlehlém prostoru
CPU	Central Processing Unit - centrální výpočetní jednotka
DPS	Deska plošného spoje
USB	Universal Serial Bus - univerzální sériová sběrnice
TTL	Transistor–Transistor Logic - tranzistorově-tranzistorová logika
RViz	Grafické prostředí frameworku ROS
CMOS	Complementary Metal–Oxide–Semiconductor - technologie pro vytváření logických obvodů
FOV	Field Of View - zorné pole
CNN	Convolutional Neural Network - konvoluční neuronová síť
FPGA	Field Programable Gate Array - programovatelné hradlové pole
GPU	Graphics Processing Unit - grafická výpočetní jednotka
SICK	Výrobce senzorů

10. Přílohy

1. Obsah CD
2. Nastavení globálního plánovače
3. Nastavení lokálního plánovače
4. Nastavení globální costmapy
5. Nastavení lokální costmapy

10.0.1. Obsah CD

Přiložené CD obsahuje elektronickou verzi diplomové práce a ukázky zdrojových kódů v následující adresářové struktuře:

/ kořenový adresář

/scripts adresář s vytvořenými a modifikovanými scripty

/config adresář s konfiguračními soubory

/launch adresář se souborem pro spuštění navigace

10.0.2. Nastavení globálního plánovače

```
## Global planner configuration
GlobalPlanner:
  use_dijkstra: true
  neutral_cost: 80
  lethal_cost: 253
  cost_factor: 0.5
```

10.0.3. Nastavení lokálního plánovače

```
## Teb local planner configuration
TebLocalPlannerROS:
  odom_topic: odometry/filtered

  ## Robot Configuration Parametres
  acc_lim_x: 0.15
  acc_lim_theta: 0.15
  max_vel_x: 0.2
  max_vel_x_backwards: 0.2
  max_vel_theta: 0.58
  min_turning_radius: 0.75
  wheelbase: 0.392
  cmd_angle_instead_rotvel: true
  max_vel_y: 0.0
  footprint_model:
    type: "two_circles"
    front_offset: 0.325
    front_radius: 0.225
    rear_offset: -0.025
    rear_radius: 0.270

  ## Goal Tolerance
  xy_goal_tolerance: 1.5
  yaw_goal_tolerance: 1
  free_goal_vel: false

  ## Trajectory Configuration Parametres
  dt_ref: 0.3
  dt_hysteresis: 0.1
  min_samples: 3
  global_plan_overwrite_orientation: true
  force_reinit_new_goal_dist: 1.0
  feasibility_check_no_poses: 3

  ## Obstackle Parameters
  min_obstacle_dist: 0.1
  include_costmap_obstacles: true
  costmap_obstacles_behind_robot_dist: 1.5
```

```
  obstacle_poses_affected: 30
  costmap_converter_spin_thread: true
  costmap_converter_rate: 5

  ## Optimization Parameters
  no_inner_iterations: 5
  no_outer_iterations: 4
  weight_max_vel_x: 2
  weight_max_vel_theta: 1
  weight_acc_lim_x: 1
  weight_acc_lim_theta: 1
  weight_kinematics_nh: 1000
  weight_kinematics_forward_drive: 10
  weight_kinematics_turning_radius: 20
  weight_optimaltime: 1
  weight_obstacle: 50
  weight_inflation: 0.1

  ## Parallel Planning in distinctive Topologies
  enable_homotopy_class_planning: false
  enable_multithreading: true
  max_number_classes: 2
  selection_cost_hysteresis: 1.0
  selection_obst_cost_scale: 1.0
  roadmap_graph_area_width: 7
  h_signature_prescaler: 0.4
  h_signature_threshold: 0.4
  obstacle_heading_threshold: 0.5
  roadmap_graph_no_samples: 15
  visualize_hc_graph: true
```

10.0.4. Nastavení globální costmapy

```
## Global costmap configuration
global_costmap:
  global_frame: /map_view
  robot_base_frame: /base_link
  map_type: costmap
  update_frequency: 1
  publish_frequency: 1.0
  transform_tolerance: 0.5
  always_send_full_costmap: true
  static_map: true
  inf_is_valid: true
  inflation_radius: 4.0
  cost_scaling_factor: 1.2
  footprint_padding: 0.05
  footprint: [[-0.3,-0.23], [0.03,-0.23], [0.3, -0.16], [0.3,0.16],
      [0.03,0.23], [-0.3, 0.23]]
```

```
  obstacle_range: 1.5
  raytrace_range: 1.8
  max_obstacle_height: 0.75
  min_obstacle_height: 0.08

  observation_sources: laser_scan_sensor1
  laser_scan_sensor1: {sensor_frame: laser, data_type: LaserScan, topic: scan,
      marking: true, clearing: true, expected_update_rate: 0.5}
```

10.0.5. Nastavení lokální costmapy

```
## Local costmap configuration
local_costmap:
  global_frame: odom
  robot_base_frame: /base_link
  map_type: costmap
  update_frequency: 4.0
  transform_tolerance: 0.5
  publish_frequency: 4.0
  static_map: false
  rolling_window: true
  width: 5
  height: 5
  resolution: 0.05
  inf_is_valid: true
  always_send_full_costmap: true
  obstacle_range: 1.5
  raytrace_range: 1.8
  footprint: [[-0.3,-0.23], [0.03,-0.23], [0.3, -0.16], [0.3,0.16],
      [0.03,0.23], [-0.3, 0.23]]
  footprint_padding: 0.08
  inflation_radius: 1.0
  cost_scaling_factor: 5.0
  max_obstacle_height: 0.75
  min_obstacle_height: 0.08

  observation_sources: laser_scan_sensor1
  laser_scan_sensor1: {sensor_frame: laser, data_type: LaserScan, topic: scan,
      marking: true, clearing: true, expected_update_rate: 0.5}
```